FAST AND COMPLETE: ENABLING COMPLETE NEURAL NETWORK VERIFICATION WITH RAPID AND MASSIVELY PARALLEL INCOMPLETE VERIFIERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Formal verification of neural networks (NNs) is a challenging NP-complete problem. Existing efficient solvers typically require the branch-and-bound (BaB) process, which splits the problem domain into subdomains and solves each subdomain using faster but weaker incomplete verifiers, such as Linear Programming (LP) on relaxed subdomains. In this paper, we propose to use the backward mode linear relaxation based perturbation analysis (LiRPA) to replace LP during the BaB process, which can be efficiently implemented on the typical machine learning accelerators such as GPUs and TPUs. However, unlike LP, LiRPA when applied naively can produce much weaker bounds and even cannot check certain conflicts during splitting, making the entire procedure incomplete after BaB. To address these challenges, we apply a fast gradient based bound tightening procedure combined with batch splits and the design of minimal usage of LP bound procedure, enabling us to effectively use LiRPA on the accelerator hardware for the challenging complete NN verification problem and significantly outperform LP based approaches. On a single GPU, we demonstrate over a magnitude speedup compared to existing LP based approaches.

1 INTRODUCTION

Even though neural networks (NNs) have demonstrated great success on various tasks, they remain susceptible to adversarial examples (Szegedy et al., 2013): imperceptible perturbations of test samples might unexpectedly cause the changes of NN predictions. Therefore, it is crucial to have formal verification of NNs such that they can be adopted in safety/security-critical settings.

In this paper we study the *complete verification* setting, where given sufficient time, the verifier should give a definite "yes/no" answer for a property under verification. Complete NN verification is generally a challenging NP-Hard problem (Katz et al., 2017). It usually requires expensive formal verification methods (such as SMT (Katz et al., 2017) and MILP solvers (Tjeng et al., 2019b)).

Recently, a framework of Branch and Bound (BaB) (Bunel et al., 2018) is widely used for efficiently verifying NNs (Lu & Kumar, 2020; Bunel et al., 2020b). In the branching step, the original verification problem will be recursively partitioned into multiple sub-problems. For each sub-problem, incomplete verification methods are then used to approximate the bound of the NN outputs and verify its truthfulness as the bounding step. BaB typically adopts Linear Program (LP) bounding procedure to provide an accurate bound approximation and feasibility checking of each split sub-problems. Specifically, LP bounding procedure will transform each sub-problem into an LP problem where the non-linearity introduced by activation functions (e.g., ReLUs) in NNs will be approximated by a linear convex relaxation. However, its high computation cost and incapability of parallelization (especially on typical hardware accelerators like GPUs or TPUs) greatly limit the performance of the existing complete BaB based verifiers.

In our paper we study how to use fast and typically weak incomplete verifiers for complete verification. Specifically, we focus on a class of incomplete verifiers using efficient bound propagation operations, referred to as linear relaxation based perturbation analysis (LiRPA) algorithms (Xu et al., 2020). Representative algorithms in this class include convex outer adversarial polytope (Wong & Kolter, 2018), CROWN (Zhang et al., 2018) and DeepPoly (Singh et al., 2019b). LiRPA algorithms

exhibit high parallelism as the bound propagation process is similar to forward or backward propagation on neural networks, which can be fully exploited by the machine learning accelerators such as GPUs and TPUs. These algorithms are sufficiently efficient, and can be even used for training certificated adversarial defense (e.g., Wong et al. (2018), Mirman et al. (2018), Wang et al. (2018a), Zhang et al. (2020)).

Although LiRPA bounds have been used for incomplete verification (especially in training certified adversarial defenses), LiRPA bounds are generally considered too loose to be useful in the complete verification setting even with BaB. As we will demonstrate later in the paper, using LiRPA bounds naively in the BaB makes the verification incomplete, so additional measures are needed to make them useful for complete verification. In fact, LiRPA methods have been used to provide intermediate layer bounds for construct LPs (Bunel et al., 2018; Lu & Kumar, 2020), however they have never been explored as the main driver in the complete verification settings. The main contributions of our paper can be summarized as:

- We show that LiRPA bounds when improved with fast gradient optimizers, can potentially outperform bounds obtained by LP verifiers, since LiRPA allows joint optimization of both intermediate layer bounds (which determine the tightness of relaxation) and output bounds, where LP can only optimize output bounds.
- We show that BaB purely using LiRPA bounds is insufficient for complete verification. To address this issue, we still need to use the LP verifier occasionally; however, the design of our algorithm makes sure that LP is only invoked when absolutely necessary, and exploits hardware parallelism when possible.
- To fully exploit the hardware parallelism on the machine learning accelerators, we use a batch splitting approach that splits on multiple neurons simultaneously, further improving the efficiency of our method.
- On a few standard and representative benchmarks, our proposed NN verification framework can outperform previous baselines by at least 10X, and in some cases (especially when the network is deeper) up to 50X.

2 Related Work

A body of recent work has shown success in efficiently verifying moderate sized neural networks (Liu et al., 2019). Most of them can be categorized into complete verifiers and incomplete verifiers. Complete verifiers guarantee to terminate either the property is proved or a violation is located. On the contrary, incomplete verifiers are much faster and usually can be massively parallelized compared to the complete ones even though no guarantee can be obtained for completeness.

Incomplete verifiers. Most of the existing incomplete verifiers verify the properties relying on sound bound propagation methods. Specifically, bound propagation methods can obtain a sound approximation of neural network output ranges by propagating the bounded input ranges through the whole neural network and relaxing the non-linear ReLU activations with convex domains (Wong & Kolter, 2018; Zhang et al., 2018; Weng et al., 2018; Wang et al., 2018b; Gehr et al., 2018; Raghunathan et al., 2018; Singh et al., 2018a; 2019b;a). Different relaxation methods will lead to different tightness of the analyzed output ranges. In general, the tighter the approximation is, the more computational and memory cost are required. We refer the readers to (Salman et al., 2019) for a comprehensive survey on bound propagation methods. Based on existing bound propagation methods, LP/MILP solvers can be customized to further tighten the approximations (Singh et al., 2018b).

Complete verifiers. Branch and Bound (BaB) is widely used in various complete verifiers. According to the branching strategies used, the BaB based methods mainly fall into two categories: (1) input domain splits (Wang et al., 2018c; Bunel et al., 2020b; Royo et al., 2019; Anderson et al., 2019) and (2) hidden activation splits (Katz et al., 2017; Bunel et al., 2018; Wang et al., 2018b; Ehlers, 2017; Lu & Kumar, 2020). Input domain split is shown effective in verifying the properties with low input dimensions while performs as poorly as incomplete verifiers on higher dimension properties. On the other hand, state-of-the-art verifiers mostly use hidden activation splits and achieve great progress in verifying harder properties. However, an LP solver is normally used for hidden activation splits and thus involves huge computational cost for each bounding propagation, making massive parallelization impossible. Apart from BaB based approaches, customized MILP (Tjeng et al., 2019a) and other LP solvers (Katz et al., 2017; Huang et al., 2017; Dutta et al., 2018) are also used for neural network verifications but do not perform as well as BaB based ones. In this paper, we consider the

Table 1: Table of Notations

Symbol	Meanings
$f(x)$	Neural network function
$l(x), u(x)$	lower and upper bounds of function $f(x)$
W^i	Weight matrix of a linear transformation layer (convolutional/fully-connected)
z^i	pre-activation output of layer i
$h(\cdot)$	Non-linear activation function
x^i	Layer $i+1$'s input, $x^i = h(z^i)$
x^0	Input of the neural network
x^o	Output of the neural network
γ_l, γ_u	Lower and upper bounds of an arbitrary variable γ
A_l^i, A_u^i	Linear coefficients of x^i in the linear lower and upper bounds of y
B_l^i, B_u^i	Bias terms of the linear lower and upper bounds between layer i inputs x^i and output y
I_+^i, I_-^i, I^i	Set of active, dead and uncertain ReLU neurons between layer i and layer $i+1$, respectively
$\mathcal{C}$	Bounded input domain

same hidden activation branching strategies used in state-of-the-art verifier (Lu & Kumar, 2020) but use massive parallel propagations with LiPRA on GPUs and seek for minimal usage of LP bounding procedure to drastically improve the verification performance.

Our paper firstly leverage the massive parallelization of weak but cheap bound propagation method on hardware accelerators to speed up complete verification. There are two similar concurrent work GPUPoly (Müller et al., 2020) and Lagrangian Decomposition (Bunel et al., 2020a) that also leverage massive parallization on GPUs for NN verification. However, GPUPoly is an extension of DeepPoly on CUDA and is *still an incomplete verifier*. Also, GPUPoly is implemented in CUDA C++, requiring manual effort for customization and gradient computation. On the contrary, our verifier is implemented in PyTorch which allows us to easily obtain the gradients and optimize the lower bound slope for LiRPA (see Section 4.1). Lagrangian Decomposition (Bunel et al., 2020a) is a complete verifier after branch and bound, and can exploit GPU for the optimization process. However, it solves an optimization problem that is more complicated than LiRPA bounds, and typically requires hundreds of iterations to converge for a single subproblem. The overall observed speedup is less than our design of using cheap and fast incomplete verifiers.

3 Background

Notations We list the notations used in our paper in Table 1.

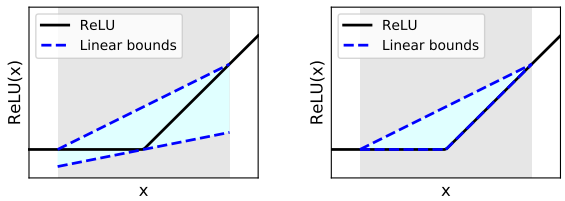
3.1 Linear Relaxation based Perturbation Analysis (LiRPA)

We start with a brief overview of linear relaxation based perturbation analysis (LiRPA). LiRPA bounds[1] maintain linear lower and upper bounds between output neurons and intermediate neurons. Such a bound can be propagated from the output neurons to previous layers in a backward manner to get a final bound between the output neurons and inputs. When a non-linear layer is encountered in this process, it will be bounded by neuron-wise linear relaxations. Finally we can get linear lower and upper bounds between the outputs and inputs of the neural network $l(x) \leq f(x) \leq u(x)$, where both $l(x)$ and $u(x)$ are linear functions in the form of $A_{l,u}^0 x + B_{l,u}^0$.

Assume that we are calculating the LiRPA bounds of a network $f(x)$, and we are at layer $i+1$, with linear bounds $A_{l \text{ or } u}^i x^i + B_{l \text{ or } u}^i$ between x^i and y. Now we will propagate the bounds backward through the ReLU activations before layer $i+1$. We can bound these ReLU neurons with linear relaxations $D_l^i X + b_l^i$ and $D_u^i X + b_u^i$. Here D_l^i and D_u^i are diagonal matrices and the elements $D_{u,j,j}^i$ and $D_{l,j,j}^i$ are the slopes of upper or lower linear relaxation of a single ReLU neuron j.

[1] We only use the backward mode LiRPA bounds (e.g., CROWN and DeepPoly) in this paper.

To choose the values of these slopes, we can divide the ReLU neurons after layer i into three groups, $I_+^i = \{j | z_{l,j}^i \geq 0\}$, $I_-^i = \{j | z_{u,j}^i < 0\}$ and $I^i = \{j | j \notin I_+^i, j \notin I_-^i\}$. For active and dead neurons in I_+^i and I_-^i, the slopes are chosen as 1 and 0. For unstable neurons in I^i, the slopes should be set such that the LiRPA bounds is a lower bound of the LP bounds. Therefore, $D_{u,j,j}^i$ should be set to be $\frac{z_{j,u}^i - z_{j,l}^i}{z_{j,u}^i}$ while $D_{l,j,j}^i$ can be any arbitrary value in $[0, 1]$ for a valid bound. This relaxation is illustrated in Figure 1a. After propagating through all the previous layers and ReLU activations, we can finally obtain the linear coefficients of final linear bounds between y and x^0 as $A = \sum_{k=0}^{K} W_k^1 D_k^1 W_k^2 D_k^2 ... D_k^n W_k^n$.



(a) Linear relaxation with an arbitrary lower bound

(b) Tightest linear convex relaxation

Figure 1: Linear relaxation of a ReLU activation

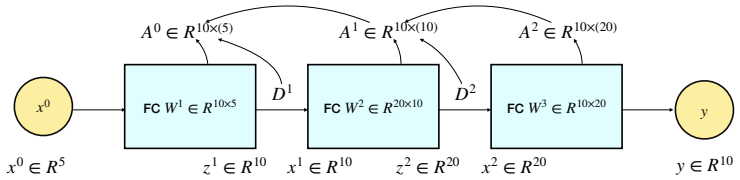


Figure 2: Backward propagation of the linear coefficient of linear bound A

We illustrate an overview of the backward propagation of bounds in Figure 2, where we show how we find linear lower and upper bounds of y with respect to x^0 in a three layer fully connected neural network.Due to the simple bound propagation rule of matrices in LiRPA bounds, this process can be efficiently implemented on hardware accelerators like GPUs as batched matrix multiplications. Propagating the LiRPA bounds can also be done automatically given a computational graph; we refer the readers to Xu et al. (2020) for a more comprehensive introduction to LiRPA based algorithms.

3.2 Branch and Bound for Neural Network Verification

Given an input x, its bounded input domain $\mathcal{C}$, and neural network $f(\cdot)$, the aim of formal verification of neural networks is to prove/disprove certain properties of neural network outputs (e.g., true label y's output is larger than other label i's). The property can be further simplified to only checking the sign of the neural network outputs by modifying its last layer weights (Wong & Kolter, 2018). Formally, neural network verification can be expressed as examining the truthfulness of Eq 1.

$$\forall x \in \mathcal{C}, f(x) \geq 0 \tag{1}$$

Due to the non-linearity and non-convexity of neural networks, finding the minimum of $f(x)$ over the bounded input domain $\mathcal{C}$ is challenging as the optimization process is generally an NP-hard problem (Katz et al., 2017). In practice, the exact minimum of $f(x)$ is usually not required to verify the property and thus a good verification strategy can save several magnitudes of time cost. BaB (Bunel et al., 2018) is usually adopted by approximating a sound lower bound $l(x)$ of the minimum value of $f(x)$ and gradually tighten the approximation by partitioning the input domain $\mathcal{C}$. Specifically, it consists of two main processes: branching and bounding. For *branching*, BaB based methods will divide the bounded input domain $\mathcal{C}$ into sub-domains $\{\mathcal{C}_i | \mathcal{C} = \bigcup_i \mathcal{C}_i\}$ by selecting different unstable ReLU units to be negative and positive cases, each defined as a new verification problem. For each sub-domain $\mathcal{C}_i$, Bab based methods will perform *bounding* to obtain a relaxed but sound lower bound $l_i(x)$ by leveraging an incomplete verifier (e.g., LP on relaxed sub-domains). A tightened global lower bound over $\mathcal{C}$ can then be obtained by taking the minimum values of generated lower bounds from all the sub-domains as $l'(x) = \min_i l_i(x)$. Branching and bounding will be performed recursively to tighten the approximated global lower bound over $\mathcal{C}$ until either (1) the approximated lower bound $l'(x)$ becomes larger than 0 and prove the property, or (2) a violation is located and disprove it.

Most of the existing incomplete verifiers mentioned in Section 2 can provide the approximated lower bound for branching. However, there is a clear trade-off between the tightness of the incomplete verifiers and the total amount of the sub-domain partitions in branching. If a cheap but weak incomplete verifier is used, then each iteration of bounding in BaB will be fast while more partitions are needed during branching. On the other hand, if the used incomplete verifier can provide a tight approximation for the lower bound, more computational cost is needed for each bounding while less partitions are needed for branching. Since each partitioned sub-domain requires to call the incomplete verifier, the total verification cost increases exponentially as more sub-domains partitioned. Therefore, most of the state-of-the-art BaB based verifiers (Bunel et al., 2020b; Lu & Kumar, 2020) seek for tight but slow incomplete verifiers by leveraging Linear Programming (LP) bounding procedure. Specifically, they transform the original optimization problem $\min_x f(x)$ where $x \in \mathcal{C}$ into a LP problem by introducing tightest convex linear relaxations over ReLU activation units (Ehlers, 2017). As shown in Figure 1b, ReLU can be replaced with the shaded triangle area in the LP construction and the approximated upper and lower for each relaxed ReLU can be computed by the existing sound bound propagation methods (Wong & Kolter, 2018).

Besides the tight approximation of the lower bound of $f(x)$, another important benefit of LP bounding procedure is its capability for checking feasibility of partitioned sub-domains. It cannot only prevent further unnecessary partitions on infeasible sub-domains but also can guarantee the completeness of whole verification process. This is because the worst-case verification cost of BaB with LP feasibility check is to split all the unstable ReLU units and then ground-truth global lower bound of $f(x)$ over $\mathcal{C}$ can be obtained (Bunel et al., 2018; Wang et al., 2018b). Note that the completeness is not guaranteed if only sound bound approximations is applied with BaB without feasibility check even though they are much faster and can be massively parallelized compared to LP bounding procedure. Thus, the huge computational cost of LP bounding procedure greatly limited the overall verification performance.

To overcome such a challenge, we propose to use massively parallelized cheap bound propagation with LiPRA on hardware accelerators. As we will discuss in the next section, efficient optimization of the cheap LiRPA bounds on GPUs allow us to achieve tight approximation on par with LP bounding procedure or even better for some cases. Also, a minimal numbers of LP bounding procedure are invoked to guarantee the completeness of our verifier.

4 Proposed Algorithm

4.1 Boosting LiRPA Bounds with Gradient Descent

As we have introduced in Sec. 3.1, the slope of lower bound of an unstable ReLU can be any arbitrary value between 0 and 1 for a valid bound. However, better values will lead to a tighter bound. Using optimal slopes for incomplete verification has been discussed in a few works (Lyu et al., 2019; Dvijotham et al., 2018), but finding the exact optimal slope for each ReLU neuron can be intractable.

In our work, we propose to use gradient descent to optimize the slopes instead of using a constant slope in the naive LiRPA implementation. In Sec.3.1, we have introduced our linear lower bound $A_l^0 x^0 + B_l^0$, and we can rewrite the linear coefficient as

$$A_l^0 = \sum_{k=0}^{K} W_k^1 D_k^1 W_k^2 D_k^2 ... D_k^n W_k^n.$$

We can then derive the global lower bound within the bounded input domain $\mathcal{C}(x^0)$ around input x_0 as

$$\min_{x' \in \mathcal{C}(x^0)} A_l^0 x' + B_l^0$$

To get a tighter lower bound, we apply gradient descent on the global lower bound, in which we can derive the gradient on a slope variable $D_{l,j,j}^i$ as

$$\frac{\partial A_l^0}{\partial D_{j,j}^i} x' + \frac{\partial B_l^0}{\partial D_{l,j,j}^i}.$$

We take the first gradient term as an example

$$\frac{\partial A_l^0}{\partial D_{l,j,j}^i} = \frac{\partial \sum_{k=0}^{K} W_k^1 D_k^1 ... W_k^i D_k^i ... D_k^n W_k^n}{\partial D_{l,j,j}^i}$$

The gradient of $D_{l,j,j}^i$ will come from two parts, one is D_k^i, which contains the element $D_{l,j,j}^i$, another is $D_k^{i'}$ where $i' > i$, which depends on the tightness of $z_u^{i'}$ and $z_l^{i'}$ and therefore depends on the lower bound slope $D_{l,j,j}^i$ on layer i.

Therefore, when we optimize $D_{l,j,j}^i$, we can also have impact on the pre-activation bounds after layer i, which means that we can indirectly optimize the bounds of intermediate layers when optimizing the final bounds of outputs. On the other hand, when we use LP to get the bounds of output layer, the upper and lower bounds of intermediate layers are precomputed and fixed. Therefore, although the LiRPA bound is a lower bound of the bound from LP bounding procedure with the same intermediate bounds, we can even get a tighter bound than LP bounding procedure after optimization. We illustrate the comparison between the bounds in Figure 3. Initially the LP bound (red) was produced using a LiRPA method (Wong & Kolter, 2018) as intermediate layer upper and lower bounds. Previous works solve this LP for each sub-domain. The blue line is the LiRPA bound on the final output neuron. By optimizing the LiRPA bound, we quickly reaches a level that produces an even tighter bound than the initial LP. The optimized LiRPA bound produced optimized intermediate layer bounds, which can be used to construct another LP with tighter relaxation (light blue), but the gap between optimized LiRPA bound and this tightened LP is not large.

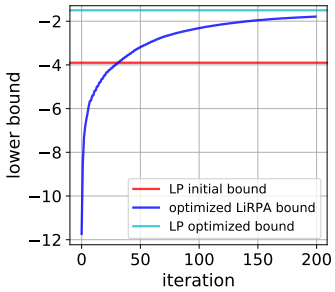


Figure 3: Compare LP optimized bound with LP initial bound and optimized LiRPA bound.

When the bound propagation process is implemented in an automatic differentiation framework such as PyTorch, projected gradient decent can be easily and efficiently obtained.

4.2 Minimal Usage of LP Bounding Procedure

Splitting an unstable ReLU neuron in LiRPA can be done by assuming a pre-activation bound to be either greater than 0 or less than 0. Ideally, if pure LiRPA is used for bounding step in BaB, we can enjoy two main benefits: (1) massive parallelization on GPU and (2) no expensive cost from LP bounding procedure. However, in terms of the LiRPA bounds, even after setting all unstable neurons to active or inactive, there can still be cases that the verification process is indecisive. In other words, the verifier with BaB and pure LiRPA without using LP bounding procedure has no guarantee to be complete. This is caused by the fact that LiRPA itself cannot detect infeasibility of split cases.

Here we provide a simple example to illustrate such a problem. Consider an NN with only two hidden node $A = B = \text{ReLU}(x)$ and one dimension input x, its output function can be written as $f(x) = A - B = \text{ReLU}(x) - \text{ReLU}(x)$. As a verification problem, we want to verify the property $f(x) \geq 0$ where $x = [-1, 1]$. Since hidden nodes A and B are exactly the same, the ground-truth output range is $f^*(x) = [0, 0]$. A complete BaB based verifier is expected to obtain that optimal bound and prove the property after split A and B together while BaB with pure LiPRA cannot guarantee that completeness. Specifically, BaB with pure LiRPA will split the original domain $x = [-1, 1]$ into four sub-domains and approximate the bound with LiRPA respectively:

(1) (feasible) sub-domain $x = [-1, 1], A > 0, B > 0$ with output $f = [x, x] - [x, x] = [0, 0]$

(2) (feasible) sub-domain $x = [-1, 1], A \leq 0, B \leq 0$ with output $f = [0, 0] - [0, 0] = [0, 0]$

(3) (infeasible) sub-domain $x = [-1, 1], A \leq 0, B > 0$ with output $f = [0, 0] - [x, x] = [-1, 1]$

(4) (infeasible) sub-domain $x = [-1, 1], A > 0, B \leq 0$ with output $f = [x, x] - [0, 0] = [-1, 1]$

Only the first two split sub-domains are feasible and therefore the ground-truth lower bound 0 can be obtained by taking the minimum of the estimated bounds from sub-domains (1) and (2). However, pure LiRPA is not able to tell the infeasibility of sub-domains (3) and (4) and thus BaB with pure

Algorithm 1 Parallel BaB

1: **Input**: net
2: global_lb, global_ub ← solve_slope(net)
3: problem ← select_prob(net)
4: probs ← [(global_lb, problem)]
5: **while** global_lb < 0 **or** probs $= \emptyset$ **do**
6: (sub_probs) ← pick_out_parallel(probs)
7: [sp_lower_1, sp_upper_1, ..., sp_lower_s, sp_upper_s] ← split(sub_probs)
8: [spb_l_1, ..., spb_u_s] ← compute_bounds_parallel(net, [sp_lower_1, ..., sp_upper_s])
9: Insert [spb_l_1, ..., spb_u_s] to probs
10: global_lb ← min{lb | (lb, prob) ∈ probs}
11: **if** global_lb not improve **then**
12: [spb_l_1, ..., spb_u_2] ← compute_bound_LP(net, [sp_lower_1, sp_upper_1])
13: Insert [spb_l_1, ..., spb_u_2] to probs
14: **end if**
15: **end while**
16: **Output** global_lb

LiRPA will report the minimum -1 got from all these four sub-domains as the global lower bound for the original input domain, ending up not being able to verify the property. Therefore, occasional LP bounding procedure is necessary to check the feasibility of split sub-domains especially when all of the hidden nodes are split.

Since LP bounding procedure is slow while LiRPA can be massively parallelized, we propose to only use LP bounding procedure when the verification process is stuck, i.e., partitioning sub-domains and estimating with LiRPA cannot further made any progress, or all unstable neurons have been split. This way, we make sure that the infeasibile sub-domains can be successfully detected by LP bounding procedure at the end while the advantage of massive parallel LiRPA on GPU is fully enjoyed. See Alg. 1 and Sec. 4.4 for a detailed description of our framework.

4.3 Batch Splits

For state-of-the-art BaB methods (Bunel et al., 2020b; Lu & Kumar, 2020), only one sub-domain can be selected for split during each branching step. Benefited from our design of using cheap LiRPA with minimal usage of LP bounding procedure, we can select a batch of sub-domains to split and propagate them in a massively parallel style. Such batching split design can introduce huge performance improvement. Given a batch size n that allows us to use up one GPU memory, we can then approximate n bounds simultaneously. This can help us grow the search tree on a single leaf by a depth of $\log_2(n)$, or split $n/2$ leaf nodes at the same time, allowing our verifier to accelerate by up to n times when applying batch splits.

4.4 Our Complete Verification Framework

The algorithm of our framework is presented in Alg. 1. Give a target model net, we first boost LiRPA bound by solving the ReLU slopes as we mentioned in Sec.4.1 to provide the best lower and upper bounds constrains at the beginning stage (Line 2) as an illustration shown in Figure 3. Then we utilize the power of GPU to parallel split and compute bounds in probs (Line 6-10). Here we improve BaBSR (Bunel et al., 2018) in a parallel manner as the decision criteria when splitting. To avoid excessive splits, global_lb is always tracked and will compute bound by LP when it dose not improve (Line 12-13). The while loop will break either global_lb < 0 which means the example is verified or probs $= \emptyset$ which means there is no more neurons to be split.

5 Experiments

In this section, we compare our verifier against the state-of-the-art ones to illustrate the effectiveness of our proposed framework. Overall, our verifier is at least 10X faster than the best verifier so far across easy, medium, and hard properties.

Setup We follow the most challenge experimental setup used in the state-of-the-art verifier GNN-ONLINE (Lu & Kumar, 2020) and BaBSR (Bunel et al., 2020b). Specifically, we evaluate on

Table 2: Verification performance of various methods on the Base model. For easy, medium and difficult level verification properties, we compare each method's average solving time, average number of branches required and the percentage of timed out properties.

	Easy			Medium			Hard		
Method	time(s)	branches	%Timeout	time(s)	branches	%Timeout	time(s)	branches	%Timeout
BaBSR	537.48	582.27	0.0	1265.40	1423.55	0.0	3005.11	2450.80	0.411
MIPPLANET	1462.24	-	0.165	1912.25	-	0.435	2172.23	-	0.462
GNN	270.93	299.72	0.002	543.12	595.40	0.004	1475.16	1077.38	0.216
GNN-ONLINE	206.43	255.81	0.002	540.15	502.51	0.001	1277.35	828.40	0.183
OURS	**57.21**	691.87	0.0	**145.24**	317.41	0.0	**275.35**	462.22	0.195

Table 3: Verification performance of various methods on Large Models. For verification properties on Wide large model and Deep large model, we compare each method's average solving time, average number of branches required and the percentage of timed out properties. Our method outperforms other methods in all aspects.

	Wide			Deep		
Method	time	branches	%Timeout	time	branches	%Timeout
BaBSR	4007.65	823.22	20.3	4016.17	456.04	0.951
MIPPLANET	5712.40	-	0.743	5122.54	-	0.608
GNN	2205.45	399.30	0.127	2105.62	212.88	0.048
GNN-ONLINE	2019.03	365.85	0.095	2214.58	277.02	0.040
OURS	**327.30**	395.45	0.118	**155.17**	199.20	0.040

CIFAR10 dataset with the same standard robustly trained convolutional neural networks, Base, Wide and Deep structures, used in (Lu & Kumar, 2020). The dataset is categorized into three different levels: easy, medium, and hard, which is generated according to the performance of BaBSR (see (Lu & Kumar, 2020) for the detailed setting of the dataset). Our experiments are conducted on one Intel E5 CPU and one Nvidia 1080TI GPU, and the parallel batch size is set to 50.

Comparisons against state-of-the-art verifiers In Table 2, we compare against the following four different methods on the same NNs: (1) BaBSR (Bunel et al., 2020b), a BaB based verifier using a ReLU-split heuristic; (2) MIPPlanet (Ehlers, 2017), a customized MIP solver for NN verification where ReLU-split is randomly selected; (3) GNN (Lu & Kumar, 2020), (4) GNN-ONLINE (Lu & Kumar, 2020), a BaB based verifier using GNN to guide the ReLU-split. The timeout threshold is 3600 seconds in easy properties and 7200 seconds for other properties. As we can see in Table 2, our approach can be up to 10X faster than baseline BaBSR. Using GNN based split can only speedup the time by a factor of about 2. Our method used the same split heuristic as BaBSR and can be improved using better method such as GNN split and we leave it for future work.

Performance comparisons on Wide and Deep model architectures The performance of the verifiers heavily depends on the size of the NN architectures. In Table 3 we show that our verifier is affected much less on wide and deep NNs compared to other state-of-the-art verifiers, and the effectiveness of our method is not affected on wider and deeper networks. Note that our method enjoys the parallel computing particularly on deep models. The accumulative solved properties with runtime is performed in Figure 4.

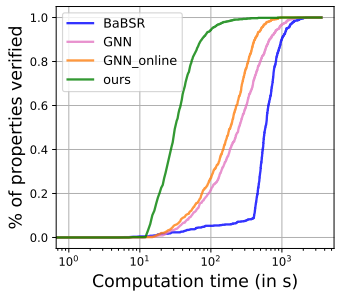


Figure 4: Cactus plots for our method and other baselines in easy properties. We plot the percentage of solved properties with runtime.

6 CONCLUSION

we propose to use fast but weak incomplete verifiers for complete verification. Specifically, we use the backward mode linear relaxation based perturbation analysis (LiRPA) to replace the general LP bounding procedure during the traditional branch and bound (BaB) process. However, unlike LP, naively applying LiRPA in BaB can produce much weaker bounds and even cannot check certain conflicts during splitting, making the entire procedure incomplete. To address these challenges, we apply a fast gradient based bound tightening procedure combined to tighten the approximated bounds for each iteration. Batch splitting and the design of minimal usage of LP bound procedure enable us to effectively use LiRPA on the accelerator hardware. On a single GPU, we demonstrate over at least 10X speedup compared to state-of-the-art verifiers across easy, medium, and hard datasets and multiple model architectures.

REFERENCES

Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. 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*, pp. 731–744, 2019.

Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Lagrangian decomposition for neural network verification. *Conference on Uncertainty in Artificial Intelligence (UAI) 2020*, 2020a.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, P Kohli, P Torr, and P Mudigonda. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020b.

Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*, pp. 4790–4799, 2018.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pp. 121–138. Springer, 2018.

Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*, 2018.

Ruediger Ehlers. 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.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18. IEEE, 2018.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pp. 3–29. Springer, 2017.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark Barrett, and Mykel J Kochenderfer. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758*, 2019.

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

Zhaoyang Lyu, Ching-Yun Ko, Zhifeng Kong, Ngai Wong, Dahua Lin, and Luca Daniel. Fastened crown: Tightened neural network robustness certificates. *arXiv preprint arXiv:1912.00574*, 2019.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pp. 3575–3583, 2018.

Christoph Müller, Gagandeep Singh, Markus Püschel, and Martin Vechev. Neural network robustness verification on gpus. *arXiv preprint arXiv:2007.10868*, 2020.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *International Conference on Learning Representations (ICLR), arXiv preprint arXiv:1801.09344*, 2018.

Vicenc Rubies Royo, Roberto Calandra, Dusan M Stipanovic, and Claire Tomlin. Fast neural network verification via shadow prices. *arXiv preprint arXiv:1902.07247*, 2019.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems 32*, pp. 9832–9842, 2019.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pp. 10825–10836, 2018a.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*, 2018b.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems*, pp. 15072–15083, 2019a.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):41, 2019b.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2013.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *ICLR*, 2019a.

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 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, 2019b. URL `https://openreview.net/forum?id=HyGIdiRqtm`.

Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Mixtrain: Scalable training of formally robust neural networks. *arXiv preprint arXiv:1811.02625*, 2018a.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pp. 6367–6377, 2018b.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1599–1614, 2018c.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pp. 5273–5282, 2018.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pp. 5283–5292, 2018.

Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. In *NIPS*, 2018.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Provable, scalable and automatic perturbation analysis on general computational graphs. *arXiv preprint arXiv:2002.12920*, 2020.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in neural information processing systems*, pp. 4939–4948, 2018.

Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020.