Journal of Machine Learning Research 25 (2024) 1-51 Submitted 1/21; Revised 11/22; Published 1/24

Scaling the Convex Barrier with Sparse Dual Algorithms

Alessandro De Palma ADEPALMA@ROBOTS.OX.AC.UK
Harkirat Singh Behl HARKIRAT@ROBOTS.OX.AC.UK
Rudy Bunel BUNEL.RUDY@GMAIL.COM
Philip H.S. Torr PHST@ROBOTS.OX.AC.UK
M. Pawan Kumar PAWAN@ROBOTS.OX.AC.UK
Department of Engineering Science
University of Oxford
Oxford OX1 3PJ

Editor: Simon Lacoste-Julien

Abstract

Tight and efficient neural network bounding is crucial to the scaling of neural network verification systems. Many efficient bounding algorithms have been presented recently, but they are often too loose to verify more challenging properties. This is due to the weakness of the employed relaxation, which is usually a linear program of size linear in the number of neurons. While a tighter linear relaxation for piecewise-linear activations exists, it comes at the cost of exponentially many constraints and currently lacks an efficient customized solver. We alleviate this deficiency by presenting two novel dual algorithms: one operates a subgradient method on a small active set of dual variables, the other exploits the sparsity of Frank-Wolfe type optimizers to incur only a linear memory cost. Both methods recover the strengths of the new relaxation: tightness and a linear separation oracle. At the same time, they share the benefits of previous dual approaches for weaker relaxations: massive parallelism, GPU implementation, low cost per iteration and valid bounds at any time. As a consequence, we can obtain better bounds than off-the-shelf solvers in only a fraction of their running time, attaining significant formal verification speed-ups.

Keywords: neural network verification, adversarial machine learning, convex optimization, branch and bound

1. Introduction

Verification requires formally proving or disproving that a given property of a neural network holds over all inputs in a specified domain. We consider properties in their canonical form (Bunel et al., 2018), which requires us to either: (i) prove that no input results in a negative output (property is true); or (ii) identify a counter-example (property is false). The search for counter-examples is typically performed by efficient methods such as random sampling of the input domain (Webb et al., 2019), or projected gradient descent (Kurakin et al., 2017; Carlini and Wagner, 2017; Madry et al., 2018). In contrast, establishing the veracity of a property requires solving a suitable convex relaxation to obtain a lower bound on the minimum output. If the lower bound is positive, the given property is true. If the bound is negative and no counter-example is found, either: (i) we make no conclusions regarding the property (incomplete verification); or (ii) we further refine the counter-example search and lower bound computation within a branch-and-bound framework until we reach a concrete conclusion (complete verification).

The main bottleneck of branch and bound is the computation of the lower bound for each node of the enumeration tree via convex optimization. While earlier works relied on off-the-shelf solvers (Ehlers, 2017; Bunel et al., 2018), it was quickly established that such an approach does not scale-up elegantly with the size of the neural network. This has motivated researchers to design specialized dual solvers (Dvijotham et al., 2020; Bunel et al., 2020a), thereby providing initial evidence that verification can be realized in practice. However, the convex relaxation considered in the dual solvers is itself very weak (Ehlers, 2017), hitting what is now commonly referred to as the "convex barrier" (Salman et al., 2019). In practice, this implies that either several properties remain undecided in incomplete verification, or take several hours to be verified exactly.

Multiple works have tried to overcome the convex barrier for piecewise linear activations (Raghunathan et al., 2018; Singh et al., 2019a). Here, we focus on the single-neuron Linear Programming (LP) relaxation by Anderson et al. (2020). Unfortunately, its tightness comes at the price of exponentially many (in the number of variables) constraints. Therefore, existing dual solvers (Dvijotham et al., 2018; Bunel et al., 2020a) are not easily applicable, limiting the scaling of the new relaxation.

We address this problem by presenting two specialized dual solvers for the relaxation by Anderson et al. (2020), which realize its full potential by meeting the following desiderata:

- Relying on an *active set* of dual variables, we present a unified dual treatment that includes both a linearly sized LP relaxation (Ehlers, 2017) and the tighter formulation. As a consequence, we obtain an inexpensive dual initializer, named Big-M, which is competitive with dual approaches on the looser relaxation (Dvijotham et al., 2018; Bunel et al., 2020a). Moreover, by dynamically extending the active set, we obtain a subgradient-based solver, named Active Set, which rapidly overcomes the convex barrier and yields much tighter bounds if a larger computational budget is available.

- The tightness of the bounds attainable by Active Set depends on the memory footprint through the size of the active set. By exploiting the properties of Frank-Wolfe style optimizers (Frank and Wolfe, 1956), we present Saddle Point, a solver that deals with the exponentially many constraints of the relaxation by Anderson et al. (2020) while only incurring a linear memory cost. Saddle Point eliminates the dependency on memory at the cost of a potential reduction of the dual feasible space, but is nevertheless very competitive with Active Set in settings requiring tight bounds.

- Both solvers are *sparse* and recover the strengths of the original primal problem (Anderson et al., 2020) in the dual domain. In line with previous dual solvers, both methods yield valid bounds at anytime, leverage convolutional network structure and enjoy *massive parallelism* within a GPU implementation, resulting in better bounds in an order of magnitude less time than off-the-shelf solvers (Gurobi Optimization, 2020). Owing to this, we show that both solvers can yield large complete verification gains compared to primal approaches (Anderson et al., 2020) and previous dual algorithms.

Code implementing our algorithms is available as part of the OVAL neural network verification framework: `https://github.com/oval-group/oval-bab`.

A preliminary version of this work appeared in the proceedings of the Ninth International Conference on Learning Representations (De Palma et al., 2021). The present article significantly extends it by:

1. Presenting Saddle Point (§5), a second solver for the relaxation by Anderson et al. (2020), which is more memory efficient than both Active Set and the original cutting plane algorithm by Anderson et al. (2020).
2. Providing a detailed experimental evaluation of the new solver, both for incomplete and complete verification.
3. Presenting an adaptive and more intuitive scheme to switch from looser to tighter bounding algorithms within branch and bound (§6.2).
4. Investigating the effect of different speed-accuracy trade-offs from the presented solvers in the context of complete verification.

2. Preliminaries: Neural Network Relaxations

We denote vectors by bold lower case letters (for example, $\mathbf{x}$) and matrices by upper case letters (for example, W). We use $\odot$ for the Hadamard product, $[\![\cdot]\!]$ for integer ranges, $\mathbb{1}_{\mathbf{a}}$ for the indicator vector on condition $\mathbf{a}$ and brackets for intervals ($[\mathbf{l}_k, \mathbf{u}_k]$) and vector or matrix entries ($\mathbf{x}[i]$ or $W[i,j]$). In addition, $\text{col}_i(W)$ and $\text{row}_i(W)$ respectively denote the i-th column and the i-th row of matrix W. Finally, given $W \in \mathbb{R}^{m\times n}$ and $\mathbf{x} \in \mathbb{R}^m$, we will employ $W \diamond \mathbf{x}$ and $W \,\square\, \mathbf{x}$ as shorthands for respectively $\sum_i \text{col}_i(W) \odot \mathbf{x}$ and $\sum_i \text{col}_i(W)^T \mathbf{x}$.

Let $\mathcal{C}$ be the network input domain. Similar to Dvijotham et al. (2018); Bunel et al. (2020a), we assume that the minimization of a linear function over $\mathcal{C}$ can be performed efficiently. For instance, this is the case for ℓ_∞ and ℓ_2 norm perturbations. Our goal is to compute bounds on the scalar output of a piecewise-linear feedforward neural network. The tightest possible lower bound can be obtained by solving the following optimization problem:

$$\begin{aligned}
\min_{\mathbf{x},\hat{\mathbf{x}}} \quad \hat{x}_n \qquad \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, && && \text{(1a)}\\
& \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} && k \in [\![0, n-1]\!], && \text{(1b)}\\
& \mathbf{x}_k = \sigma\left(\hat{\mathbf{x}}_k\right) && k \in [\![1, n-1]\!], && \text{(1c)}
\end{aligned}$$

where the activation function $\sigma\left(\hat{\mathbf{x}}_k\right)$ is piecewise-linear, $\hat{\mathbf{x}}_k, \mathbf{x}_k \in \mathbb{R}^{n_k}$ denote the outputs of the k-th linear layer (fully-connected or convolutional) and activation function respectively, W_k and $\mathbf{b}_k$ denote its weight matrix and bias, n_k is the number of activations at layer k. We will focus on the ReLU case ($\sigma\left(\mathbf{x}\right) = \max\left(\mathbf{x}, 0\right)$), as common piecewise-linear functions can be expressed as a composition of ReLUs (Bunel et al., 2020b).

Problem (1) is non-convex due to the activation function's non-linearity, that is, due to constraint (1c). As solving it is NP-hard (Katz et al., 2017), it is commonly approximated by a convex relaxation (see §7). The quality of the corresponding bounds, which is fundamental in verification, depends on the tightness of the relaxation. Unfortunately, tight relaxations usually correspond to slower bounding procedures. We first review a popular ReLU relaxation in §2.1. We then consider a tighter one in §2.2.

2.1 Planet Relaxation

The so-called Planet relaxation (Ehlers, 2017) has enjoyed widespread use due to its amenability to efficient customized solvers (Dvijotham et al., 2018; Bunel et al., 2020a) and is the "relaxation of choice" for many works in the area (Salman et al., 2019; Singh et al., 2019b; Bunel et al., 2020b; Balunovic and Vechev, 2020; Lu and Kumar, 2020). Here, we describe it in its non-projected form $\mathcal{M}_k$, corresponding to the LP relaxation of the Big-M Mixed Integer Programming (MIP) formulation (Tjeng et al., 2019). Applying $\mathcal{M}_k$ to problem (1) results in the following linear program:

$$\begin{aligned}
\min_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}} \quad \hat{x}_n \quad \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C} \\
& \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} && k \in [\![0, n-1]\!], \\
& \left.\begin{array}{l}
\mathbf{x}_k \geq \hat{\mathbf{x}}_k, \quad \mathbf{x}_k \leq \hat{\mathbf{u}}_k \odot \mathbf{z}_k, \\
\mathbf{x}_k \leq \hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k \odot (1 - \mathbf{z}_k), \\
(\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \times [\mathbf{0},\ \mathbf{1}]
\end{array}\right\} := \mathcal{M}_k && k \in [\![1, n-1]\!],
\end{aligned} \tag{2}$$

where $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ and $\mathbf{l}_k, \mathbf{u}_k$ are *intermediate bounds* respectively on pre-activation variables $\hat{\mathbf{x}}_k$ and post-activation variables $\mathbf{x}_k$. These constants play an important role in the structure of $\mathcal{M}_k$ and, together with the relaxed binary constraints on $\mathbf{z}$, define box constraints on the variables. We detail how to compute intermediate bounds in appendix E. Projecting out auxiliary variables $\mathbf{z}$ results in the Planet relaxation (cf. appendix B.1 for details), which replaces (1c) by its convex hull.

Problem (2), which is linearly-sized, can be easily solved via commercial black-box LP solvers (Bunel et al., 2018). This does not scale-up well with the size of the neural network, motivating the need for specialized solvers. Customized dual solvers have been designed by relaxing constraints (1b), (1c) (Dvijotham et al., 2018) or replacing (1c) by the Planet relaxation and employing Lagrangian Decomposition (Bunel et al., 2020a). Both approaches result in bounds very close to optimality for problem (2) in only a fraction of the runtime of off-the-shelf solvers.

2.2 A Tighter Relaxation

A much tighter approximation of problem (1) than the Planet relaxation (§2.1) can be obtained by representing the convex hull of the composition of constraints (1b) and (1c) rather than the convex hull of constraint (1c) alone. A formulation of this type was recently introduced by Anderson et al. (2020). In order to represent the interaction between $\mathbf{x}_k$ and all possible subsets of the activations of the previous layer, the formulation relies on a number of auxiliary matrices, which we will now introduce. Weight matrices are masked entry-wise via I_k, binary masks belonging to the following set: $2^{W_k} = \{0,1\}^{n_k \times n_{k-1}}$. We define $\mathcal{E}_k := 2^{W_k} \setminus \{0,1\}$, to exclude the all-zero and all-one masks, which we treat separately (see §2.2.1). In addition, the formulation requires bounds on the subsets of $\mathbf{x}_{k-1}$ selected via the masking. In order to represent these bounds, the formulation exploits matrices $\check{L}_{k-1}, \check{U}_{k-1} \in \mathbb{R}^{n_k \times n_{k-1}}$, which are also masked via I_k and computed via interval arithmetic as follows:

$$\begin{aligned}
\check{L}_{k-1}[i,j] &= \mathbf{l}_{k-1}[j]\mathbb{1}_{W_k[i,j]\geq 0} + \mathbf{u}_{k-1}[j]\mathbb{1}_{W_k[i,j]<0}, \\
\check{U}_{k-1}[i,j] &= \mathbf{u}_{k-1}[j]\mathbb{1}_{W_k[i,j]\geq 0} + \mathbf{l}_{k-1}[j]\mathbb{1}_{W_k[i,j]<0}.
\end{aligned}$$

The new representation results in the following primal problem:

$$\begin{aligned}
&\min_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}} \ \hat{x}_n \ \text{ s.t.} \\
&\quad \mathbf{x}_0 \in \mathcal{C} \\
&\quad \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1} && k \in [\![0, n-1]\!]\,, \\
&\quad \left.\begin{aligned}
&(\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in \mathcal{M}_k \\
&\mathbf{x}_k \leq \begin{pmatrix} (W_k \odot I_k)\,\mathbf{x}_{k-1} + \mathbf{z}_k \odot \mathbf{b}_k + \\ -\big(W_k \odot I_k \odot \check{L}_{k-1}\big) \diamond (1 - \mathbf{z}_k) + \\ + \big(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\big) \diamond \mathbf{z}_k \end{pmatrix} \ \forall I_k \in \mathcal{E}_k
\end{aligned}\right\} := \mathcal{A}_k && k \in [\![1, n-1]\!]\,.
\end{aligned} \tag{3}$$

Both $\mathcal{M}_k$ and $\mathcal{A}_k$ yield valid MIP formulations for problem (1) when imposing integrality constraints on $\mathbf{z}$. However, the LP relaxation of $\mathcal{A}_k$ will yield tighter bounds. In the worst case, this tightness comes at the cost of exponentially many constraints: one for each $I_k \in \mathcal{E}_k$. On the other hand, given a set of primal assignments $(\mathbf{x}, \mathbf{z})$ that are not necessarily feasible for problem (3), one can efficiently compute the most violated constraint (if any) at that point. Denoting by $\mathcal{A}_{\mathcal{E},k} = \mathcal{A}_k \setminus \mathcal{M}_k$ the exponential family of constraints, the mask associated to the most violated constraint in $\mathcal{A}_{\mathcal{E},k}$ can be computed in linear-time (Anderson et al., 2020) as:

$$I_k[i,j] = \mathbb{1}^T_{\big((1-\mathbf{z}_k[i])\odot \check{L}_{k-1}[i,j] + \mathbf{z}_k[i]\odot \check{U}_{k-1}[i,j] - \mathbf{x}_{k-1}[j]\big)W_k[i,j] \geq 0}. \tag{4}$$

The most violated constraint in $\mathcal{A}_k$ is then obtained by comparing the constraint violation from the output of oracle (4) to those from the constraints in $\mathcal{M}_k$.

2.2.1 Pre-activation Bounds

Set $\mathcal{A}_k$ defined in problem (3) slightly differs from the original formulation of Anderson et al. (2020), as the latter does not exploit pre-activation bounds $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ within the exponential family. In particular, constraints $\mathbf{x}_k \leq \hat{\mathbf{u}}_k \odot \mathbf{z}_k$, and $\mathbf{x}_k \leq \hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k \odot (1 - \mathbf{z}_k)$, which we treat via $\mathcal{M}_k$, are replaced by looser counterparts. While this was implicitly addressed in practical applications (Botoeva et al., 2020), not doing so has a strong negative effect on bound tightness, possibly to the point of yielding looser bounds than problem (2). In appendix F, we provide an example in which this is the case and extend the original derivation by Anderson et al. (2020) to recover $\mathcal{A}_k$ as in problem (3).

2.2.2 Cutting Plane Algorithm

Owing to the exponential number of constraints, problem (3) cannot be solved as it is. As outlined by Anderson et al. (2020), the availability of a linear-time separation oracle (4) offers a natural primal *cutting plane* algorithm, which can then be implemented in off-the-shelf solvers: solve the Big-M LP (2), then iteratively add the most violated constraints from $\mathcal{A}_k$ at the optimal solution. When applied to the verification of small neural networks via off-the-shelf MIP solvers, this leads to substantial gains with respect to the looser Big-M relaxation (Anderson et al., 2020).

3. A Dual Formulation for the Tighter Relaxation

Inspired by the success of dual approaches on looser relaxations (Bunel et al., 2020a; Dvijotham et al., 2020), we show that the formal verification gains by Anderson et al. (2020) (see §2.2) scale to larger networks if we solve the tighter relaxation in the dual space. Due to the particular structure of the relaxation, a customized solver for problem (3) needs to meet a number of requirements.

Fact 1 *In order to replicate the success of previous dual algorithms on looser relaxations, we need a solver for problem* (3) *with the following properties: (i)* sparsity*: a memory cost linear in the number of network activations in spite of exponentially many constraints, (ii)* tightness*: the bounds should reflect the quality of those obtained in the primal space, (iii)* anytime*: low cost per iteration and valid bounds at each step.*

The anytime requirement motivates dual solutions: any dual assignment yields a valid bound due to weak duality. Unfortunately, as shown in appendix A, neither of the two dual derivations by Bunel et al. (2020a); Dvijotham et al. (2018) readily satisfy all desiderata at once. Therefore, we need a completely different approach. Starting from primal (3), we relax all constraints in $\mathcal{A}_k$ except box constraints (see §2.1). n order to simplify notation, we employ dummy variables $\boldsymbol{\alpha}_0 = 0$, $\boldsymbol{\beta}_0 = 0$, $\boldsymbol{\alpha}_n = I$, $\boldsymbol{\beta}_n = 0$, obtaining the following dual problem (derivation in appendix D):

$$\begin{aligned}
&\max_{(\boldsymbol{\alpha},\boldsymbol{\beta})\geq 0} d(\boldsymbol{\alpha},\boldsymbol{\beta}) \qquad \text{where:} \qquad d(\boldsymbol{\alpha},\boldsymbol{\beta}) := \min_{\mathbf{x},\mathbf{z}} \ \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}),\\
&\mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}) = \left[\begin{array}{l} \sum_{k=1}^{n-1} \mathbf{b}_k^T \boldsymbol{\alpha}_k - \sum_{k=0}^{n-1} \boldsymbol{f}_k(\boldsymbol{\alpha},\boldsymbol{\beta})^T \mathbf{x}_k - \sum_{k=1}^{n-1} \boldsymbol{g}_k(\boldsymbol{\beta})^T \mathbf{z}_k \\ + \sum_{k=1}^{n-1} \left(\sum_{I_k \in \mathcal{E}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \,\square\, \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1}^T (\hat{\mathbf{l}}_k - \mathbf{b}_k) \right) \end{array}\right] \qquad (5)\\
&\text{s.t.} \qquad \mathbf{x}_0 \in \mathcal{C}, \qquad (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] \qquad k \in [\![1, n-1]\!],
\end{aligned}$$

where functions $\boldsymbol{f}_k, \boldsymbol{g}_k$ are defined as follows:

$$\begin{aligned}
\boldsymbol{f}_k(\boldsymbol{\alpha},\boldsymbol{\beta}) &= \ \boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - \textstyle\sum_{I_k} \boldsymbol{\beta}_{k,I_k} + \sum_{I_{k+1}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}},\\
\boldsymbol{g}_k(\boldsymbol{\beta}) &= \left[\begin{array}{l} \sum_{I_k \in \mathcal{E}_k} \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k \\ + \sum_{I_k \in \mathcal{E}_k} \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond \boldsymbol{\beta}_{k,I_k} + \sum_{I_k \in \mathcal{E}_k} \boldsymbol{\beta}_{k,I_k} \odot \mathbf{b}_k. \end{array}\right] \qquad (6)
\end{aligned}$$

We employ $\sum_{I_k}$ as a shorthand for $\sum_{I_k \in 2^{W_k}}$. Both functions therefore include sums over an exponential number of $\boldsymbol{\beta}_{k,I_k}$ variables.

This is again a challenging problem: the exponentially many constraints in the primal (3) are associated to an exponential number of variables, as $\boldsymbol{\beta} = \{\boldsymbol{\beta}_{k,I_k} \forall I_k \in 2^{W_k}, k \in [\![1, n-1]\!]\}$. Nevertheless, we show that the requirements of Fact 1 can be met by operating on restricted versions of the dual. We present two specialized algorithms for problem (5): one considers only a small active set of dual variables (§4), the other restricts the dual domain while exploiting the sparsity of Frank-Wolfe style iterates (§5). Both algorithms are sparse, anytime and yield bounds reflecting the tightness of the new relaxation.

We conclude this section by pointing out that, as stated in §2, the only assumption for $\mathcal{C}$ in problem (5) is that it allows for efficient linear minimization. This is the case for both ℓ_∞ and ℓ_2 norm perturbations, which are hence supported by our solvers.

Algorithm 1 Active Set

```
 1: function ACTIVESET_COMPUTE_BOUNDS(Problem (5))
 2:   Initialize duals $\boldsymbol{\alpha}^0, \boldsymbol{\beta}_0^0, \boldsymbol{\beta}_1^0$ using Algorithm (3)
 3:   Set $\boldsymbol{\beta}^0_{k,I_k} = 0, \forall\ I_k \in \mathcal{E}_k$
 4:   $\mathcal{B} = \emptyset$
 5:   for nb_additions do
 6:     for $t \in [\![0, T-1]\!]$ do
 7:       $\mathbf{x}^{*,t}, \mathbf{z}^{*,t} \in \text{argmin}_{\mathbf{x},\mathbf{z}}\, \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^t, \boldsymbol{\beta}^t_{\mathcal{B}})$ using (8),(9)        ▷ inner minimization
 8:       if $t \leq$ nb_vars_to_add then
 9:         For each layer $k$, add output of (4) called at $(\mathbf{x}^*, \mathbf{z}^*)$ to $\mathcal{B}_k$        ▷ active set extension
10:       end if
11:       $\boldsymbol{\alpha}^{t+1}, \boldsymbol{\beta}^{t+1}_{\mathcal{B}} \leftarrow (\boldsymbol{\alpha}^t, \boldsymbol{\beta}^t) + H(\nabla_{\boldsymbol{\alpha}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}), \nabla_{\boldsymbol{\beta}_{\mathcal{B}}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}))$        ▷ supergradient step, using (10)
12:       $\boldsymbol{\alpha}^{t+1}, \boldsymbol{\beta}^{t+1}_{\mathcal{B}} \leftarrow \max(\boldsymbol{\alpha}^{t+1}, 0), \max(\boldsymbol{\beta}^{t+1}_{\mathcal{B}}, 0)$        ▷ dual projection
13:     end for
14:   end for
15:   return $\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^T, \boldsymbol{\beta}^T_{\mathcal{B}})$
16: end function
```

4. Active Set

We present Active Set (Algorithm 1), a supergradient-based solver that operates on a small active set of dual variables $\boldsymbol{\beta}_{\mathcal{B}}$. Starting from the dual of problem (2), Active Set iteratively adds variables to $\boldsymbol{\beta}_{\mathcal{B}}$ and solves the resulting reduced version of problem (5). We first describe our solver on a fixed $\boldsymbol{\beta}_{\mathcal{B}}$ (§4.1) and then outline how to iteratively modify the active set (§4.2).

4.1 Solver

We want to solve a version of problem (5) for which $\mathcal{E}_k$, the exponentially-sized set of I_k masks for layer k, is restricted to some constant-sized set[1] $\mathcal{B}_k \subseteq \mathcal{E}_k$, with $\mathcal{B} = \cup_{k\in[\![1,n-1]\!]}\mathcal{B}_k$. By keeping $\mathcal{B} = \emptyset$, we recover a novel dual solver for the Big-M relaxation (2) (explicitly described in appendix B), which is employed as initialization. Setting $\boldsymbol{\beta}_{k,I_k} = 0, \forall\ I_k \in \mathcal{E}_k \setminus \mathcal{B}_k$ in (6), (5) and removing these from the formulation, we obtain:

$$\boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}}) = \left[\begin{array}{l} \boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - \sum_{I_k \in \mathcal{B}_k \cup \{0,1\}} \boldsymbol{\beta}_{k,I_k}, \\ +\sum_{I_{k+1} \in \mathcal{B}_{k+1} \cup \{0,1\}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}} \end{array}\right.$$

$$\boldsymbol{g}_{\mathcal{B},k}(\boldsymbol{\beta}_{\mathcal{B}}) = \left[\begin{array}{l} \sum_{I_k \in \mathcal{B}_k} \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k \\ + \sum_{I_k \in \mathcal{B}_k} \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond \boldsymbol{\beta}_{k,I_k} + \sum_{I_k \in \mathcal{B}_k} \boldsymbol{\beta}_{k,I_k} \odot \mathbf{b}_k, \end{array}\right.$$

1. As dual variables $\boldsymbol{\beta}_{k,I_k}$ are indexed by I_k, $\mathcal{B} = \cup_k \mathcal{B}_k$ implicitly defines an active set of variables $\boldsymbol{\beta}_{\mathcal{B}}$.

along with the reduced dual problem:

$$
\begin{aligned}
&\max_{(\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B})\geq 0} d_\mathcal{B}(\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B}) \qquad \text{where:} \qquad d_\mathcal{B}(\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B}) := \min_{\mathbf{x},\mathbf{z}} \ \mathcal{L}_\mathcal{B}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B}),\\
&\mathcal{L}_\mathcal{B}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B}) = \left[\begin{array}{l} \sum_{k=1}^{n-1} \mathbf{b}_k^T \boldsymbol{\alpha}_k - \sum_{k=0}^{n-1} \boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B})^T \mathbf{x}_k - \sum_{k=1}^{n-1} \boldsymbol{g}_{\mathcal{B},k}(\boldsymbol{\beta}_\mathcal{B})^T \mathbf{z}_k \\ + \sum_{k=1}^{n-1} \left(\sum_{I_k \in \mathcal{B}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \,\square\, \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1}^T (\hat{\mathbf{l}}_k - \mathbf{b}_k) \right) \end{array}\right. \qquad (7)\\
&\text{s.t.} \qquad \mathbf{x}_0 \in \mathcal{C}, \qquad (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] \qquad k \in [\![1, n-1]\!].
\end{aligned}
$$

We can maximize $d_\mathcal{B}(\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B})$, which is concave and non-smooth, via projected supergradient ascent or variants thereof, such as Adam (Kingma and Ba, 2015). In order to obtain a valid supergradient, we need to perform the inner minimization over the primals. Thanks to the structure of problem (7), the optimization decomposes over the layers. For $k \in [\![1, n-1]\!]$, we can perform the minimization in closed-form by driving the primals to their upper or lower bounds depending on the sign of their coefficients:

$$
\mathbf{x}_k^* = \mathbb{1}_{\boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B})\geq 0} \odot \hat{\mathbf{u}}_k + \mathbb{1}_{\boldsymbol{f}_{\mathcal{B},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B})< 0} \odot \hat{\mathbf{l}}_k, \qquad \mathbf{z}_k^* = \mathbb{1}_{\boldsymbol{g}_{\mathcal{B},k}(\boldsymbol{\beta}_\mathcal{B})\geq 0} \odot \mathbf{1}. \qquad (8)
$$

The subproblem corresponding to $\mathbf{x}_0$ is different, as it involves a linear minimization over $\mathbf{x}_0 \in \mathcal{C}$:

$$
\mathbf{x}_0^* \in \ \operatorname{argmin}_{\mathbf{x}_0} \quad \boldsymbol{f}_{\mathcal{B},0}(\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B})^T \mathbf{x}_0 \qquad \text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}. \qquad (9)
$$

We assumed in §2 that (9) can be performed efficiently. We refer the reader to Bunel et al. (2020a) for descriptions of the minimization when $\mathcal{C}$ is a ℓ_∞ or ℓ_2 ball, as common for adversarial examples.

Given $(\mathbf{x}^*, \mathbf{z}^*)$ as above, the supergradient of $d_\mathcal{B}(\boldsymbol{\alpha},\boldsymbol{\beta}_\mathcal{B})$ is a subset of the one for $d(\boldsymbol{\alpha},\boldsymbol{\beta})$, given by:

$$
\begin{aligned}
&\nabla_{\boldsymbol{\alpha}_k} d(\boldsymbol{\alpha},\boldsymbol{\beta}) = W_k \mathbf{x}_{k-1}^* + \mathbf{b}_k - \mathbf{x}_k^*, \qquad \nabla_{\boldsymbol{\beta}_{k,0}} d(\boldsymbol{\alpha},\boldsymbol{\beta}) = \mathbf{x}_k^* - \mathbf{z}_k^* \odot \hat{\mathbf{u}}_k,\\
&\nabla_{\boldsymbol{\beta}_{k,1}} d(\boldsymbol{\alpha},\boldsymbol{\beta}) = \mathbf{x}_k^* - \left(W_k \mathbf{x}_{k-1}^* + \mathbf{b}_k\right) + (1 - \mathbf{z}_k^*) \odot \hat{\mathbf{l}}_k, \qquad (10)\\
&\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha},\boldsymbol{\beta}) = \left(\begin{array}{l} \mathbf{x}_k^* - (W_k \odot I_k)\, \mathbf{x}_{k-1}^* + \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k^*) + \\ -\mathbf{z}_k^* \odot \mathbf{b}_k + \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k^* \end{array}\right) I_k \in \mathcal{B}_k,
\end{aligned}
$$

for each $k \in [\![1, n-1]\!]$ (dual "variables" $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_n, \boldsymbol{\beta}_0, \boldsymbol{\beta}_n$ are constants employed to simplify the notation: see appendix D). At each iteration, after taking a step in the supergradient direction, the dual variables are projected to the non-negative orthant by clipping negative values.

4.2 Extending the Active Set

We initialize the dual (5) with a tight bound on the Big-M relaxation by solving for $d_\emptyset(\boldsymbol{\alpha},\boldsymbol{\beta}_\emptyset)$ in (7) (appendix B). To satisfy the tightness requirement in Fact 1, we then need to include constraints (via their Lagrangian multipliers) from the exponential family of $\mathcal{A}_k$ into $\mathcal{B}_k$. Our goal is to tighten them as much as possible while keeping the active set small to save memory and compute. The active set strategy is defined by a *selection criterion* for the I_k^* to be added[2] to $\mathcal{B}_k$, and the *frequency* of addition. In practice, *we add the variables maximising the entries of supergradient* $\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha},\boldsymbol{\beta})$ *after a fixed number of dual iterations.* We now provide motivation for both choices.

2. adding a single I_k^* mask to $\mathcal{B}_k$ extends $\boldsymbol{\beta}_\mathcal{B}$ by n_k variables: one for each neuron at layer k.

4.2.1 Selection Criterion

The selection criterion needs to be computationally efficient. Thus, we proceed greedily and focus only on the immediate effect at the current iteration. Let us map a restricted set of dual variables $\boldsymbol{\beta}_\mathcal{B}$ to a set of dual variables $\boldsymbol{\beta}$ for the full dual (5). We do so by setting variables not in the active set to 0: $\boldsymbol{\beta}_{\bar{\mathcal{B}}} = 0$, and $\boldsymbol{\beta} = \boldsymbol{\beta}_\mathcal{B} \cup \boldsymbol{\beta}_{\bar{\mathcal{B}}}$. Then, for each layer k, we add the set of variables $\boldsymbol{\beta}_{k,I_k^*}$ maximising the corresponding entries of the supergradient of the full dual problem (5), excluding those pertaining to $\mathcal{M}_k$:

$$\boldsymbol{\beta}_{k,I_k^*} \in \operatorname*{argmax}_{\boldsymbol{\beta}_{k,I_k} \in \, \boldsymbol{\beta}_k \setminus \boldsymbol{\beta}_{\emptyset,k}} \{\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})^T \mathbf{1}\}. \tag{11}$$

Therefore, we use the subderivatives as a proxy for short-term improvement on the full dual objective $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$. Under a primal interpretation, our selection criterion involves a call to the separation oracle (4) by Anderson et al. (2020).

Proposition 1 *$\boldsymbol{\beta}_{k,I_k^*}$ as defined in equation* (11) *represents the Lagrangian multipliers associated to the most violated constraints from $\mathcal{A}_{\mathcal{E},k}$ at $(\mathbf{x}^*, \mathbf{z}^*) \in \operatorname{argmin}_{\mathbf{x},\mathbf{z}} \mathcal{L}_\mathcal{B}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_\mathcal{B})$, the primal minimiser of the current restricted Lagrangian.*

Proof The result can be obtained by noticing that $\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ (by definition of Lagrangian multipliers) quantifies the corresponding constraint's violation at $(\mathbf{x}^*, \mathbf{z}^*)$. Therefore, maximizing $\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ will amount to maximizing constraint violation. We demonstrate analytically that the process will, in fact, correspond to a call to oracle (4). Recall the definition of $\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ in equation (8), which applies beyond the current active set:

$$\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \begin{pmatrix} \mathbf{x}_k^* - (W_k \odot I_k)\, \mathbf{x}_{k-1}^* + \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k^*) + \\ -\mathbf{z}_k^* \odot \mathbf{b}_k + \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k^* \end{pmatrix} \quad I_k \in \mathcal{E}_k.$$

We want to compute $I_k^* \in \operatorname{argmax}_{I_k \in \mathcal{E}_k} \{\nabla_{\boldsymbol{\beta}_{k,I_k}} d(\boldsymbol{\alpha}, \boldsymbol{\beta})^T \mathbf{1}\}$, that is:

$$I_k^* \in \operatorname*{argmax}_{I_k \in \mathcal{E}_k} \begin{pmatrix} \mathbf{x}_k^* - (W_k \odot I_k)\, \mathbf{x}_{k-1}^* + \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k^*) + \\ -\mathbf{z}_k^* \odot \mathbf{b}_k + \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k^* \end{pmatrix}^T \mathbf{1}.$$

By removing the terms that do not depend on I_k, we obtain:

$$\max_{I_k \in \mathcal{E}_k} \begin{pmatrix} -(W_k \odot I_k)\, \mathbf{x}_{k-1}^* + \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k^*) + \\ + \left(W_k \odot I_k \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k^* \end{pmatrix}^T \mathbf{1}.$$

Let us denote the i-th row of W_k and I_k by $\boldsymbol{w}_{i,k}$ and $\boldsymbol{i}_{i,k}$, respectively, and define $\mathcal{E}_k[i] = 2^{\boldsymbol{w}_{i,k}} \setminus \{0, 1\}$. The optimization decomposes over each such row: we thus focus on the optimization problem for the supergradient's i-th entry. Collecting the mask, we get:

$$\max_{\boldsymbol{i}_{i,k} \in \mathcal{E}_k[i]} \; \sum_j \left(\left((1 - \mathbf{z}_k^*[i]) \odot \check{L}_{k-1}[i,j] + \mathbf{z}_k^*[i] \odot \check{U}_{k-1}[i,j] - \mathbf{x}_{k-1}^*[i] \right) W_k[i,j] \right) I_k[i,j] \; .$$

As the solution to the problem above is obtained by setting $I_k^*[i,j] = 1$ if its coefficient is positive and $I_k^*[i,j] = 0$ otherwise, we can see that the optimal I_k corresponds to calling oracle (4) by Anderson et al. (2020) on $(\mathbf{x}^*, \mathbf{z}^*)$. Hence, in addition to being the mask associated to $\boldsymbol{\beta}_{k,I_k^*}$, the variable set maximising the supergradient, I_k^* corresponds to the most violated constraint from $\mathcal{A}_{\mathcal{E},k}$ at $(\mathbf{x}^*, \mathbf{z}^*)$. ∎

4.2.2 Frequency of Addition

Finally, we need to decide the frequency at which to add variables to the active set.

Fact 2 *Assume we obtained a dual solution* $(\boldsymbol{\alpha}^{\dagger}, \boldsymbol{\beta}_{\mathcal{B}}^{\dagger}) \in \operatorname{argmax} d_{\mathcal{B}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$ *using Active Set on the current* $\mathcal{B}$. *Then* $(\mathbf{x}^{*}, \mathbf{z}^{*}) \in \operatorname{argmin}_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^{\dagger}, \boldsymbol{\beta}_{\mathcal{B}}^{\dagger})$ *is not in general an optimal primal solution for the primal of the current variable-restricted dual problem (Sherali and Choi, 1996).*

The primal of $d_{\mathcal{B}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$ (restricted primal) is the problem obtained by setting $\mathcal{E}_k \leftarrow \mathcal{B}_k$ in problem (3). While the primal cutting plane algorithm by Anderson et al. (2020) calls the separation oracle (4) at the optimal solution of the current restricted primal, Fact 2 shows that our selection criterion leads to a different behaviour even at dual optimality for $d_{\mathcal{B}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$. Therefore, as we have no theoretical incentive to reach (approximate) subproblem convergence, we add variables after a fixed tunable number of supergradient iterations. Furthermore, we can add more than one variable "at once" by running the oracle (4) repeatedly for a number of iterations.

We conclude this section by pointing out that, provided the algorithm is run to dual convergence on each variable-restricted dual problem (7), primal optima can be recovered by suitable modifications of the optimization routine (Sherali and Choi, 1996). Then, if the dual variables corresponding to the most violated constraint at the primal optima are added to $\mathcal{B}_k$, Active Set mirrors the primal cutting plane algorithm, guaranteeing convergence to the solution of problem (5). In practice, as the main advantage of dual approaches (Dvijotham et al., 2018; Bunel et al., 2020a) is their ability to quickly achieve tight bounds (rather than formal optimality), we rely on the heuristic strategy in Algorithm 1.

5. Saddle Point

For the Active Set solver (§4), we only consider settings in which $\boldsymbol{\beta}_{\mathcal{B}}$ is composed of a (small) constant number of variables. In fact, both its memory cost and time complexity per iteration are proportional to the cardinality of the active set. This mirrors the properties of the primal cutting algorithm by Anderson et al. (2020), for which memory and runtime will increase with the number of added constraints. As a consequence, the tightness of the attainable bounds will depend both on the computational budget and on the available memory. We remove the dependency on memory by presenting Saddle Point (Algorithm 2), a Frank-Wolfe type solver. By restricting the dual feasible space, Saddle Point is able to deal with all the exponentially many variables from problem (5), while incurring only a linear memory cost. We first describe the rationale behind the reduced dual domain (§5.1), then proceed to describe solver details (§5.2).

5.1 Sparsity via Sufficient Statistics

In order to achieve sparsity (Fact 1) without resorting to active sets, it is crucial to observe that all the appearances of $\boldsymbol{\beta}$ variables in $\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$, the Lagrangian of the full dual (5), can be traced back to the following linearly-sized *sufficient statistics*:

$$\boldsymbol{\zeta}_k(\boldsymbol{\beta}_k) = \begin{bmatrix} \sum_{I_k} \boldsymbol{\beta}_{k,I_k} \\ \sum_{I_{K+1}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}} \\ \sum_{I_k \in \mathcal{E}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1} \odot (\hat{\mathbf{l}}_k - \mathbf{b}_k) \\ \sum_{I_k \in \mathcal{E}_k} (W_k \odot I_k \odot \check{U}_{k-1}) \diamond \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,0} \odot (\hat{\mathbf{u}}_k - \mathbf{b}_k) \end{bmatrix}. \tag{12}$$

Therefore, by designing a solver that only requires access to $\boldsymbol{\zeta}_k(\boldsymbol{\beta}_k)$ rather than to single $\boldsymbol{\beta}_k$ entries, we can incur only a linear memory cost.

In order for the resulting algorithm to be computationally efficient, we need to meet the anytime requirement in Fact 1 with a low cost per iteration. Let us refer to the evaluation of a neural network at a given input point $\mathbf{x}_0$ as a forward pass, and the backpropagation of a gradient through the network as a backward pass.

Fact 3 *The full dual objective $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ can be computed at the cost of a backward pass over the neural network if sufficient statistics $\boldsymbol{\zeta}(\boldsymbol{\beta}) = \cup_{k\in[\![1,n-1]\!]} \boldsymbol{\zeta}_k(\boldsymbol{\beta}_k)$ have been pre-computed.*

Proof If $\boldsymbol{\zeta}(\boldsymbol{\beta})$ is up to date, the Lagrangian $\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ can be evaluated using a single backward pass: this can be seen by replacing the relevant entries of (12) in equations (6) and (5). Similarly to gradient backpropagation, the bottleneck of the Lagrangian computation is the layer-wise use of transposed linear operators over the $\boldsymbol{\alpha}$ dual variables. The minimization of the Lagrangian over primals can then be computed in linear time (less than the cost of a backward pass) by using equations (8), (9) with $\mathcal{B}_k = \mathcal{E}_k$ for each layer k. ■

From Fact 3, we see that the full dual can be efficiently evaluated via $\boldsymbol{\zeta}(\boldsymbol{\beta})$. On the other hand, in the general case, $\boldsymbol{\zeta}(\boldsymbol{\beta})$ updates have an exponential time complexity. Therefore, we need a method that updates the sufficient statistics while computing a minimal number of terms of the exponentially-sized sums in (12). In other words, we need *sparse updates* in the $\boldsymbol{\beta}$ variables. With this goal in mind, we consider methods belonging to the Frank-Wolfe family (Frank and Wolfe, 1956), whose iterates are known to be sparse (Jaggi, 2013). In particular, we now illustrate that sparse updates can be obtained by applying the Saddle-Point Frank-Wolfe (SP-FW) algorithm by Gidel et al. (2017) to a suitably modified version of problem (5). Details of SP-FW and the solver resulting from its application are then presented in §5.2.

Fact 4 *Dual problem* (5) *can be seen as a bilinear saddle point problem. By limiting the dual feasible region to a compact set, a dual optimal solution for this domain-restricted problem can be obtained via SP-FW (Gidel et al., 2017). Moreover, a valid lower bound to* (3) *can be obtained at anytime by evaluating $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ at the current dual point from SP-FW.*

We make the feasible region of problem (5) compact by capping the cumulative price for constraint violations at some constants $\boldsymbol{\mu}$. In particular, we bound the ℓ_1 norm for the sets of $\boldsymbol{\beta}$ variables associated to each neuron. As the ℓ_1 norm is well-known to be sparsity inducing

Algorithm 2 Saddle Point

```
1: function SADDLEPOINT_COMPUTE_BOUNDS(Problem (5))
2:     Initialize duals $\boldsymbol{\alpha}^0, \boldsymbol{\beta}^0_{\mathcal{B}}$ using algorithm (3) or algorithm (1)
3:     Set $\boldsymbol{\beta}^0_{\bar{\mathcal{B}}} = 0$, $\boldsymbol{\beta}^0 = \boldsymbol{\beta}^0_{\mathcal{B}} \cup \boldsymbol{\beta}^0_{\bar{\mathcal{B}}}$, and replace $\boldsymbol{\beta}^0$ by its sufficient statistics $\boldsymbol{\zeta}(\boldsymbol{\beta}^0_k)$ using (12)
4:     Initialize primals $\mathbf{x}^0, \mathbf{z}^0$ according to §C.2
5:     Set price caps $\boldsymbol{\mu}$ heuristically as outlined in §C.1
6:     for $t \in [\![0, T-1]\!]$ do
7:         $\bar{\mathbf{x}}^t, \bar{\mathbf{z}}^t \leftarrow$ using (8),(9) with $\mathcal{B}_k = \mathcal{E}_k$          ▷ compute primal conditional gradient
8:         $\bar{\boldsymbol{\alpha}}^t, \boldsymbol{\zeta}(\bar{\boldsymbol{\beta}}^t) \leftarrow$ (14), (15) + (12)          ▷ compute dual conditional gradient
9:         $\mathbf{x}^{t+1}, \mathbf{z}^{t+1}, \boldsymbol{\alpha}^{t+1}, \boldsymbol{\zeta}(\boldsymbol{\beta}^{t+1}) = (1-\gamma_t)[\mathbf{x}^t, \mathbf{z}^t, \boldsymbol{\alpha}^t, \boldsymbol{\zeta}(\boldsymbol{\beta}^t)] + \gamma_t[\bar{\mathbf{x}}^t, \bar{\mathbf{z}}^t, \bar{\boldsymbol{\alpha}}^t, \boldsymbol{\zeta}(\bar{\boldsymbol{\beta}}^t)]$
10:    end for
11:    return $\min_{\mathbf{x},\mathbf{z}} \mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^T, \boldsymbol{\zeta}(\boldsymbol{\beta}^T))$
12: end function
```

(Candès et al., 2008), our choice reflects the fact that, in general, only a fraction of the $\mathcal{A}_k$ constraints will be active at the optimal solution. Denoting by $\triangle(\boldsymbol{\mu}) = \cup_{k\in[\![1,n-1]\!]}\triangle_k(\boldsymbol{\mu}_k)$ the resulting dual domain, we obtain domain-restricted dual $\max_{(\boldsymbol{\alpha},\boldsymbol{\beta})\in\triangle(\boldsymbol{\mu})} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$, which can be written as the following saddle point problem:

$$\begin{aligned}
&\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \min_{\mathbf{x},\mathbf{z}} \ \mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\
&\text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}, \\
&\qquad (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] \qquad\qquad k \in [\![1, n-1]\!], \\
&\qquad \left.\begin{array}{l} \boldsymbol{\alpha}_k \in [\mathbf{0}, \boldsymbol{\mu}_{\alpha,k}] \\ \boldsymbol{\beta}_k \geq \mathbf{0},\ \|\boldsymbol{\beta}_k\|_1 \leq \boldsymbol{\mu}_{\beta,k} \end{array}\right\} := \triangle_k(\boldsymbol{\mu}_k) \qquad k \in [\![1, n-1]\!],
\end{aligned} \tag{13}$$

where $\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ was defined in equation (5) and $\|\cdot\|_1$ denotes the ℓ_1 norm. Frank-Wolfe type algorithms move towards the vertices of the feasible region. Therefore, the shape of $\triangle_k(\boldsymbol{\mu}_k)$ is key to the efficiency of $\boldsymbol{\zeta}_k$ updates. In our case, $\triangle_k(\boldsymbol{\mu}_k)$ is the Cartesian product of a box constraint on $\boldsymbol{\alpha}_k$ and n_k exponentially-sized simplices: one for each set $\boldsymbol{\beta}_k[i] = \{\boldsymbol{\beta}_{k,\text{row}_i(I_k)}[i]\ \forall\ \text{row}_i(I_k) \in 2^{\text{row}_i(W_k)}\}$. As a consequence, each vertex of $\triangle_k(\boldsymbol{\mu}_k)$ is sparse in the sense that at most $n_k + 1$ variables out of exponentially many will be non-zero. In order for the resulting solver to be useful in practice, we need to efficiently select a vertex towards which to move: we show in section 5.2 that our choice for $\triangle_k(\boldsymbol{\mu}_k)$ allows us to recover the linear-time primal oracle (4) by Anderson et al. (2020).

Before presenting the technical details of our Saddle Point solver, it remains to comment on the consequences of the dual space restriction on the obtained bounds. Let us define $d^*_{\boldsymbol{\mu}} = \max_{(\boldsymbol{\alpha},\boldsymbol{\beta})\in\triangle(\boldsymbol{\mu})} d(\boldsymbol{\alpha}, \boldsymbol{\beta})$, the optimal value of the restricted dual problem associated to saddle point problem (13). Value $d^*_{\boldsymbol{\mu}}$ is attained at the dual variables from a saddle point of problem (13). As we restricted the dual feasible region, $d^*_{\boldsymbol{\mu}}$ will in general be smaller than the optimal value of problem (5). However, owing to the monotonicity of $d^*_{\boldsymbol{\mu}}$ over $\boldsymbol{\mu}$ and the concavity of $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$, we can make sure $\triangle(\boldsymbol{\mu})$ contains the optimal dual variables by running a binary search on $\boldsymbol{\mu}$. In practice, we heuristically determine the values of $\boldsymbol{\mu}$ from our dual initialization procedure (see appendix C.1).

5.2 Solver

Algorithms in the Frank-Wolfe family proceed by taking convex combinations between the current iterate and a vertex of the feasible region. This ensures feasibility of the iterates without requiring projections. For SP-FW (Gidel et al., 2017), the convex combination is performed at once, with the same coefficient, for both primal and dual variables.

In the general case, denoting primal variables as $\mathbf{x}$, and dual variables as $\mathbf{y}$, each iteration of the SP-FW algorithm proceeds as follows: first, we compute the vertex $[\bar{\mathbf{x}}, \bar{\mathbf{y}}]$ towards which we take a step (*conditional gradient*). This is done by maximizing the inner product between the gradient and the variables over the feasible region for the dual variables, and by minimizing the inner product between the gradient and the variables over the feasible region for the primal variables. This operation is commonly referred to as the linear maximization oracle for dual variables, and linear minimization oracle for primal variables. Second, a step size $\gamma_t \in [0, 1]$ is determined according to the problem specification. Finally, the current iterate is updated as $[\mathbf{x}, \mathbf{y}] \leftarrow (1 - \gamma_t)[\mathbf{x}, \mathbf{y}] + \gamma_t[\bar{\mathbf{x}}, \bar{\mathbf{y}}]$. We will now provide details for the instantiation of SP-FW in the context of problem (13), along with information concerning the solver initialization.

While Saddle Point relies on a primal-dual method operating on problem (13), our main goal is to compute anytime bounds to problem (3). As explained in §3, this is typically achieved in the dual domain. Therefore, as per Fact 4, we discard the primal variables from SP-FW and use the current dual iterate to evaluate $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$ from problem (5).

5.2.1 Conditional Gradient Computations

Due to the bilinearity of $\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$, the computation of the conditional gradient for the primal variables coincides with the inner minimization in equations (8)-(9) with $\mathcal{B}_k = \mathcal{E}_k \ \forall\ k \in [\![1, n-1]\!]$.

Similarly to the primal variables, the linear maximization oracle for the dual variables decomposes over the layers. The gradient of the Lagrangian over the duals, $\nabla_{\boldsymbol{\alpha},\boldsymbol{\beta}}\ \mathcal{L}$, is given by the supergradient in equation (10) if $\mathcal{B}_k = \mathcal{E}_k$ and the primal minimiser $(\mathbf{x}^*, \mathbf{z}^*)$ is replaced by the primals at the current iterate. As dual variables $\boldsymbol{\alpha}$ are box constrained, the linear maximization oracle will drive them to their lower or upper bounds depending on the sign of their gradient. Denoting conditional gradients by bars, for each $k \in [\![1, n-1]\!]$:

$$\bar{\boldsymbol{\alpha}}_k = \boldsymbol{\mu}_{\alpha,k} \odot \mathbb{1}_{(W_k \mathbf{x}_{k-1} + \mathbf{b}_k - \mathbf{x}_k) \geq 0}. \tag{14}$$

The linear maximization for the exponentially many $\boldsymbol{\beta}_k$ variables is key to the solver's efficiency and is carried out on a Cartesian product of simplex-shaped sub-domains (see definition of $\triangle_k(\boldsymbol{\mu}_k)$ in (13)). Therefore, conditional gradient $\bar{\boldsymbol{\beta}}_k$ can be non-zero only for the entries associated to the largest gradients of each simplex sub-domain. For each $k \in [\![1, n-1]\!]$, we have:

$$\bar{\boldsymbol{\beta}}_k = \left\{ \begin{array}{l} \bar{\boldsymbol{\beta}}_{k,I_k^\dagger} = \boldsymbol{\mu}_{\beta,k} \odot \mathbb{1}_{\nabla_{\boldsymbol{\beta}_{k,I_k^\dagger}} \mathcal{L}(\mathbf{x},\mathbf{z},\boldsymbol{\alpha},\boldsymbol{\beta}) \geq 0} \\ \bar{\boldsymbol{\beta}}_{k,I_k} = \mathbf{0} \quad \forall\ I_k \in 2^{W_k} \setminus I_k^\dagger \end{array} \right\}, \tag{15}$$
$$\text{where:} \quad \boldsymbol{\beta}_{k,I_k^\dagger} \in \underset{\boldsymbol{\beta}_{k,I_k} \in \boldsymbol{\beta}_k}{\operatorname{argmax}} \left\{ \nabla_{\boldsymbol{\beta}_{k,I_k}}\ \mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})^T \mathbf{1} \right\}.$$

We can then efficiently represent $\bar{\boldsymbol{\beta}}_k$ through sufficient statistics as $\bar{\boldsymbol{\zeta}}_k = \boldsymbol{\zeta}_k(\bar{\boldsymbol{\beta}}_k)$, which will require the computation of a single term of the sums in (12).

Proposition 2 $\boldsymbol{\beta}_{k,I_k^\dagger}$ *as defined in* (15) *represents the Lagrangian multipliers associated to the most violated constraints from* $\mathcal{A}_k$ *at* $(\mathbf{x}, \mathbf{z})$*, the current SP-FW primal iterate. Moreover, the conditional gradient* $\bar{\boldsymbol{\beta}}_k$ *can be computed at the cost of a single call to the linear-time oracle* (4) *by Anderson et al. (2020).*

Proof Let us define $\boldsymbol{\beta}_{k,I_k^*}$ as:

$$\boldsymbol{\beta}_{k,I_k^*} \in \underset{\boldsymbol{\beta}_{k,I_k} \in\ \boldsymbol{\beta}_k \setminus \boldsymbol{\beta}_{\emptyset,k}}{\operatorname{argmax}} \{\nabla_{\boldsymbol{\beta}_{k,I_k}} \mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})^T \mathbf{1}\}.$$

Proceeding as the proof of proposition 1, with $(\mathbf{x}, \mathbf{z})$ in lieu of $(\mathbf{x}^*, \mathbf{z}^*)$, we obtain that $\boldsymbol{\beta}_{k,I_k^*}$ is the set of Lagrangian multipliers for the most violated constraint from $\mathcal{A}_{\mathcal{E},k}$ at $(\mathbf{x}, \mathbf{z})$ and can be computed through the oracle (4) by Anderson et al. (2020).

Then, $\boldsymbol{\beta}_{k,I_k^\dagger}$ is computed as:

$$\boldsymbol{\beta}_{k,I_k^\dagger} \in \underset{\boldsymbol{\beta}_{k,I_k} \in\ \{\boldsymbol{\beta}_{k,I_k^*},\ \boldsymbol{\beta}_{k,0},\ \boldsymbol{\beta}_{k,1}\}}{\operatorname{argmax}} \nabla_{\boldsymbol{\beta}_{k,I_k}} \mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})^T \mathbf{1}.$$

As pointed out in the proof of proposition 1, the dual gradients of the Lagrangian correspond (by definition of Lagrangian multiplier) to constraint violations. Hence, $\boldsymbol{\beta}_{k,I_k^\dagger}$ is associated to the most violated constraint in $\mathcal{A}_k$. ■

5.2.2 Convex Combinations

The $(t+1)$-th SP-FW iterate will be given by a convex combination of the t-th iterate and the current conditional gradient. Due to the linearity of $\boldsymbol{\zeta}(\boldsymbol{\beta})$, we can perform the operation via sufficient statistics. Therefore, all the operations of Saddle Point occur in the linearly-sized space of $(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\zeta}(\boldsymbol{\beta}))$:

$$[\mathbf{x}^{t+1}, \mathbf{z}^{t+1}, \boldsymbol{\alpha}^{t+1}, \boldsymbol{\zeta}(\boldsymbol{\beta}^{t+1})] = (1-\gamma_t)[\mathbf{x}^t, \mathbf{z}^t, \boldsymbol{\alpha}^t, \boldsymbol{\zeta}(\boldsymbol{\beta}^t)] + \gamma_t[\bar{\mathbf{x}}^t, \bar{\mathbf{z}}^t, \bar{\boldsymbol{\alpha}}^t, \bar{\boldsymbol{\zeta}}(\boldsymbol{\beta}^t)],$$

where, for our bilinear objective, SP-FW prescribes $\gamma_t = \frac{1}{1+t}$ (Gidel et al., 2017, section 5).

Extensions of SP-FW, such as its away or pairwise step variants, require a worst-case memory cost that is linear in the number of iterations. In other words, as for Active Set, the attainable tightness would depend on the available memory, voiding one of the main advantages of Saddle Point (we provide empirical evidence of its memory efficiency in §8.2.2). Furthermore, the worst-case memory cost would increase more rapidly than for Active Set, which infrequently adds a few variables to $\mathcal{B}$ (in our experiments, $\mathcal{B}$ contains at most 7 variables per neuron: see §8.2). Finally, due to the bilinearity of the objective, these SP-FW variants do not correspond to an improved convergence rate (Gidel et al., 2017).

5.2.3 Initialization

As for the Active Set solver (§4), dual variables can be initialized via supergradient ascent on the set of dual variables associated to the Big-M relaxation (cf. appendix B). Additionally, if the available memory permits it, the initialization can be tightened by running Active Set (algorithm 1) for a small fixed number of iterations.

We mirror this strategy for the primal variables, which are initialized by performing subgradient descent on the primal view of saddle point problem (13). Analogously to the dual case, the primal view of problem (13) can be restricted to the Big-M relaxation for a cheaper initialization. Our primal initialization strategy is detailed in in appendix C.2.

6. Implementation Details, Technical Challenges

In this section, we present details concerning the implementation of our solvers. In particular, we first outline our parallelization scheme and the need for a specialized convolutional operator (§6.1), then describe how to efficiently employ our solvers within branch and bound (§6.2).

6.1 Parallelism, Masked Forward/Backward Passes

Analogously to previous dual algorithms (Dvijotham et al., 2018; Bunel et al., 2020a), our solvers can leverage the massive parallelism offered by modern GPU architectures in three different ways. First, we execute in parallel the computations of lower and upper bounds relative to all the neurons of a given layer. Second, in complete verification, we can batch over the different Branch and Bound (BaB) subproblems. Third, as most of our solvers rely on standard linear algebra operations employed during the forward and backward passes of neural networks, we can exploit the highly optimized implementations commonly found in modern deep learning frameworks.

An exception are what we call "masked" forward and backward passes. Writing convolutional operators in the form of their equivalent linear operator (as done in previous sections, see §2), masked passes take the following form:

$$(W_k \odot I_k)\, \mathbf{a}_k, \quad (W_k \odot I_k)^T\, \mathbf{a}_{k+1},$$

where $\mathbf{a}_k \in \mathbb{R}^{n_k}, \mathbf{a}_{k+1} \in \mathbb{R}^{n_{k+1}}$. Both operators are needed whenever dealing with constraints from $\mathcal{A}_k$. In fact, they appear in both Saddle Point (for instance, in the sufficient statistics (12)) and Active Set (see equations (7), (10)), except for the dual initialization procedure based on the easier Big-M problem (2).

Masked passes can be easily implemented for fully connected layers via Hadamard products. However, a customized lower-level implementation is required for a proper treatment within convolutional layers. In fact, from the high-level perspective, masking a convolutional pass corresponds to altering the content of the convolutional filter while it is being slided through the image. Details of our implementation can be found in appendix G.

6.2 Stratified Bounding for Branch and Bound

In complete verification, we aim to solve the non-convex problem (1) directly, rather than an approximation like problem (3). In order to do so, we rely on branch and bound, which operates by dividing the problem domain into subproblems (branching) and bounding the local minimum over those domains. The lower bound on the minimum is computed via a bounding algorithm, such as our solvers (§4, §5). The upper bound, instead, can be obtained by evaluating the network at an input point produced by the lower bound computation.[3] Any domain that cannot contain the global lower bound is pruned away, whereas the others are kept and branched over. The graph describing branching relations between sub-problems is referred to as the enumeration tree. As tight bounding is key to pruning the largest possible number of domains, the bounding algorithm plays a crucial role. Moreover, it usually acts as a computational bottleneck for branch and bound (Lu and Kumar, 2020).

In general, tighter bounds come at a larger computational cost. The overhead can be linked to the need to run dual iterative algorithms for more iterations, or to the inherent complexity of tighter relaxations like problem (3). For instance, such complexity manifests itself in the masked passes described in appendix G, which increase the cost per iteration of Active Set and Saddle Point with respect to algorithms operating on problem (2). These larger costs might negatively affect performance on easier verification tasks, where a small number of domain splits with loose bounds suffices to verify the property. Therefore, as a general complete verification system needs to be efficient regardless of problem difficulty, we employ a *stratified* bounding system within branch and bound. Specifically, we devise a simple adaptive heuristic to determine whether a given subproblem is "easy" (therefore looser bounds are sufficient) or whether it is instead preferable to rely on tighter bounds.

Given a bounding algorithm a, let us denote its lower bound for subproblem s as $l_a(s)$. Assume two different bounding algorithms, a_l and a_t, are available: one inexpensive yet loose, the other tighter and more costly. At the root r of the branch and bound procedure, we estimate $l_{a_t - a_l} = l_{a_t}(r) - l_{a_l}(r)$, the extent to which the lower bounds returned by a_l can be tightened by a_t. While exploring the enumeration tree, we keep track of the lower bound increase from parent to child (that is, after splitting the subdomain) through an exponential moving average. We write $i(s)$ for the average parent-to-child tightening until subproblem s. Then, under the assumption that each subtree is complete, we can estimate $|s|_{a_l}$ and $|s|_{a_t}$, the sizes of the enumeration subtrees rooted at s that would be generated by each bounding algorithm. Recall that, for verification problems the canonical form (Bunel et al., 2018), subproblems are discarded when their lower bound is positive. Given p, the parent of subproblem s, we perform the estimation as: $|s|_{a_l} = 2^{\frac{-l_{a_l}(p)}{i(s)}+1} - 1$, $|s|_{a_t} = 2^{\frac{-\left(l_{a_l}(p)+l_{a_t-a_l}\right)}{i(s)}+1} - 1$. Then, relying on c_{a_t/a_l}, a rough estimate of the relative overhead of running a_t over a_l, we mark the subtree rooted at s as hard if the reduction in tree size from using a_t exceeds its overhead. That is, if $\frac{|s|_{a_l}}{|s|_{a_t}} > c_{a_t/a_l}$, the lower bound for s and its children will be computed via algorithm a_t rather than a_l.

3. For subgradient-type methods like Active Set, we evaluate the network at $\mathbf{x}_0^{*,T}$ (see algorithm 1), while for Frank-Wolfe-type methods like Saddle Point at $\mathbf{x}_0^T$ (see algorithm 2). Running the bounding algorithm to get an upper bound would result in a much looser bound, as it would imply having an upper bound on a version of problem (1) with maximization instead of minimization.

7. Related Work

In addition to those described in §2, many other relaxations have been proposed in the literature. In fact, all bounding methods are equivalent to solving some convex relaxation of a neural network. This holds for conceptually different ideas such as bound propagation (Gowal et al., 2018), specific dual assignments (Wong and Kolter, 2018), dual formulations based on Lagrangian Relaxation (Dvijotham et al., 2018) or Lagrangian Decomposition (Bunel et al., 2020a). The degree of tightness varies greatly: from looser relaxations associated to closed-form methods (Gowal et al., 2018; Weng et al., 2018; Wong and Kolter, 2018) to tighter formulations based on Semi-Definite Programming (SDP) (Raghunathan et al., 2018).

The speed of closed-form approaches results from simplifying the triangle-shaped feasible region of the Planet relaxation (§2.1) (Singh et al., 2018; Wang et al., 2018). On the other hand, tighter relaxations are more expressive than the linearly-sized LP by Ehlers (2017). The SDP formulation by Raghunathan et al. (2018) can represent interactions between activations in the same layer. Similarly, Singh et al. (2019a) tighten the Planet relaxation by considering the convex hull of the union of polyhedra relative to k ReLUs of a given layer at once. Alternatively, tighter LPs can be obtained by considering the ReLU together with the affine operator before it: standard MIP techniques (Jeroslow, 1987) lead to a formulation that is quadratic in the number of variables (see appendix F.2). The relaxation by Anderson et al. (2020) detailed in §2.2 is a more convenient representation of the same set.

By projecting out the auxiliary $\mathbf{z}$ variables, Tjandraatmadja et al. (2020) recently introduced another formulation equivalent to the one by Anderson et al. (2020), with half as many variables and a linear factor more constraints compared to what described in §2.2. Therefore, the relationship between the two formulations mirrors the one between the Planet and Big-M relaxations (see appendix B.1). Our dual derivation and solvers can be adapted to operate on the projected relaxations. Furthermore, the formulation by Tjandraatmadja et al. (2020) allows for a propagation-based method ("FastC2V"). However, such an algorithm tackles only two constraints per neuron at once and might hence yield looser bounds than the Planet relaxation. In this work, we are interested in designing solvers that can operate on strict subsets of the feasible region from problem (2).

Specialized dual solvers significantly improve in bounding efficiency with respect to off-the-shelf solvers for both LP (Bunel et al., 2020a) and SDP formulations (Dvijotham et al., 2020). Therefore, the design of similar solvers for other tight relaxations is an interesting line of future research. We contribute with two specialized dual solvers for the relaxation by Anderson et al. (2020). In what follows, we demonstrate empirically that by meeting the requirements of Fact 1 we can obtain large incomplete and complete verification improvements.

8. Experiments

We empirically demonstrate the effectiveness of our methods under two settings. First, we assess the speed and quality of our bounds compared to other bounding algorithms on incomplete verification (§8.2). Then, we examine whether our speed-accuracy trade-offs pay off within branch and bound (§8.3). The implementation of our algorithms, available at `https://github.com/oval-group/oval-bab` as part of the OVAL neural network verification framework, is based on Pytorch (Paszke et al., 2017).

8.1 Experimental Setting

We compare both against dual iterative methods and Gurobi, which we use as gold standard for LP solvers. The latter is employed for the following two baselines:

- **Gurobi Planet** means solving the Planet Ehlers (2017) relaxation of the network (a version of problem (2) for which $\mathbf{z}$ have been projected out).
- **Gurobi cut** starts from the Big-M relaxation and adds constraints from $\mathcal{A}_k$ in a cutting-plane fashion, as the original primal algorithm by Anderson et al. (2020).

Both Gurobi-based methods make use of LP incrementalism (warm-starting) when possible. In the experiments of §8.2, where each image involves the computation of 9 different output upper bounds, we warm-start each LP from the LP of the previous neuron. For "Gurobi 1 cut", which involves two LPs per neuron, we first solve all Big-M LPs, then proceed with the LPs containing a single cut.

In addition, our experimental analysis comprises the following dual iterative methods:

- **BDD+**, the recent proximal-based solver by Bunel et al. (2020a), operating on a Lagrangian Decomposition dual of the Planet relaxation.
- **Active Set** denotes our supergradient-based solver for problem (3), described in §4.
- **Saddle Point**, our Frank-Wolfe-based solver for problem (3) (as presented in §5).
- By keeping $\mathcal{B} = \emptyset$, Active Set reduces to **Big-M**, a solver for the non-projected Planet relaxation (appendix B), which is employed as dual initializer to both Active Set and Saddle Point.
- **AS-SP** is a version of Saddle Point whose dual initialization relies on a few iterations of Active Set rather than on the looser Big-M solver, hence combining both our dual approaches.

As we operate on the same data sets employed by Bunel et al. (2020a), we omit both their supergradient-based approach and the one by Dvijotham et al. (2018), as they both perform worse than BDD+ (Bunel et al., 2020a). For the same reason, we omit cheaper (and looser) methods, like interval propagation (Gowal et al., 2018) and the one by Wong and Kolter (2018). In line with previous bounding algorithms (Bunel et al., 2020a), we employ Adam updates (Kingma and Ba, 2015) for supergradient-type methods due to their faster empirical convergence. While dual iterative algorithms are specifically designed to take advantage of GPU acceleration (see §6.1), we additionally provide a CPU implementation of our solvers in order to complement the comparison with Gurobi-based methods.

Unless otherwise stated, experiments were run under the following setup: Ubuntu 16.04.2 LTS, on a single Nvidia Titan Xp GPU, except those based on Gurobi and the CPU version of our solvers. The latter were run on i7-6850K CPUs, utilising 4 cores for the incomplete verification experiments, and 6 cores for the more demanding complete verification setting.

8.2 Incomplete Verification

We evaluate the efficacy of our bounding algorithms in an incomplete verification setting by upper bounding the vulnerability to adversarial perturbations (Szegedy et al., 2014), measured as the difference between the logits associated to incorrect classes and the one corresponding to the ground truth, on the CIFAR-10 test set (Krizhevsky and Hinton, 2009). If the upper bound is negative, we can certify the network's robustness to adversarial perturbations.

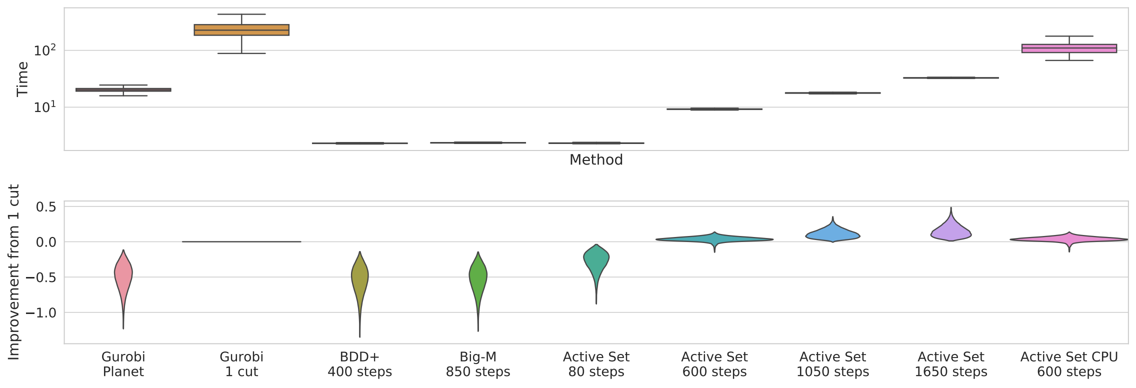
8.2.1 Speed-Accuracy Trade-Offs

Here, we replicate the experimental setting from Bunel et al. (2020a). The networks correspond to the small network architecture from Wong and Kolter (2018), and to the "Wide" architecture, also employed for complete verification experiments in §8.3.1, found in Table 1. Due to the additional computational cost of bounds obtained via the tighter relaxation (3), we restricted the experiments to the first 2567 CIFAR-10 test set images for the experiments on the SGD-trained network (Figures 1, 2), and to the first 4129 images for the network trained via the method by Madry et al. (2018) (Figures 9, 10).

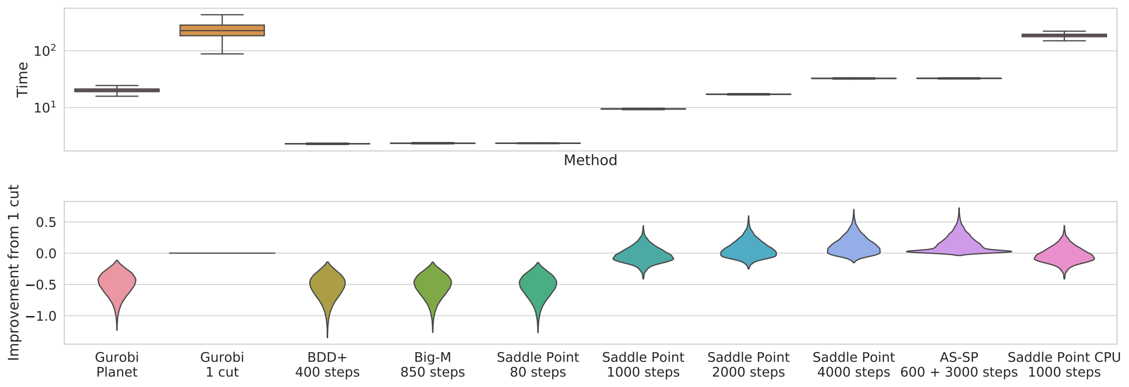
Here, we present results for a network trained via standard SGD and cross entropy loss, with no modification to the objective for robustness. Perturbations for this network lie in a ℓ_∞ norm ball with radius $\epsilon_{ver} = 1.125/255$ (which is hence lower than commonly employed radii for robustly trained networks). In appendix H, we provide additional CIFAR-10 results on an adversarially trained network using the method by Madry et al. (2018), and on MNIST (LeCun et al., 1998), for a network trained with the verified training algorithm by Wong and Kolter (2018).

Solver hyper-parameters were tuned on a small subset of the CIFAR-10 test set. BDD+ is run with the hyper-parameters found by Bunel et al. (2020a) on the same data sets, for both incomplete and complete verification. For all supergradient-based methods (Big-M, Active Set), we employed the Adam update rule (Kingma and Ba, 2015), which showed stronger empirical convergence. For Big-M, replicating the findings by Bunel et al. (2020a) on their supergradient method, we linearly decrease the step size from 10^{-2} to 10^{-4}. Active Set is initialized with 500 Big-M iterations, after which the step size is reset and linearly scaled from 10^{-3} to 10^{-6}. We found the addition of variables to the active set to be effective before convergence: we add variables every 450 iterations, without re-scaling the step size again. Every addition consists of 2 new variables (see algorithm 1) and we cap the maximum number of cuts to 7. This was found to be a good compromise between fast bound improvement and computational cost. For Saddle Point, we use $1/(t+10)$ as step size to stay closer to the initialization points. The primal initialization algorithm (see appendix C.2) is run for 100 steps on the Big-M variables, with step size linearly decreased from 10^{-2} to 10^{-5}.

Figure 1 shows the distribution of runtime and the bound improvement with respect to Gurobi cut for the SGD-trained network. For Gurobi cut, we only add the single most violated cut from $\mathcal{A}_k$ per neuron, due to the cost of repeatedly solving the LP. We tuned BDD+ and Big-M, the dual methods operating on the weaker relaxation (2), to have the same average runtime. They obtain bounds comparable to Gurobi Planet in one order less time. Initialized from 500 Big-M iterations, at 600 iterations, Active Set already achieves



(a) Speed-accuracy trade-offs of Active Set for different iteration ranges and computing devices.



(b) Speed-accuracy trade-offs of Saddle Point for different iteration ranges and computing devices.

Figure 1: Upper bounds to the adversarial vulnerability for the SGD-trained network from Bunel et al. (2020a). Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better, the width at a given value represents the proportion of problems for which this is the result. On average, both Active Set and Saddle Point achieve bounds tighter than Gurobi 1 cut with a smaller runtime.

better bounds on average than Gurobi cut in around $1/20^{th}$ of the time. With a computational budget twice as large (1050 iterations) or four times as large (1650 iterations), the bounds significantly improve over Gurobi cut in a fraction of the time. Similar observations hold for Saddle Point which, especially when using fewer iterations, also exhibits a larger variance in terms of bounds tightness. In appendix H, we empirically demonstrate that the tightness of the Active Set bounds is strongly linked to our active set strategy (see §4.2). Remarkably, even if our solvers are specifically designed to take advantage of GPU acceleration, executing them on CPU proves to be strongly competitive with Gurobi cut. Active Set produces better bounds in less time for the benchmark of Figure 1, while Saddle Point yields comparable speed-accuracy trade-offs.

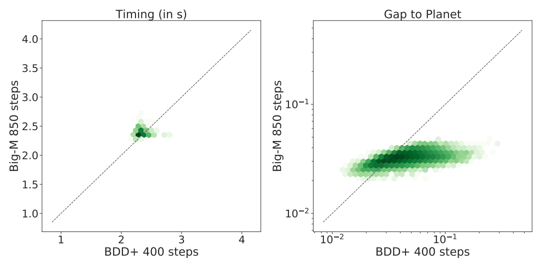
Figure 2 shows pointwise comparisons for the less expensive methods from Figure 1, on the same data. Figure 2(a) shows the gap to the (Gurobi) Planet bound for BDD+

and our Big-M solver. Surprisingly, our Big-M solver is competitive with BDD+, achieving on average better bounds than BDD+, in the same time. Figure 2(b) shows the improvement over Planet bounds for Big-M compared to those of few (80) Active Set and Saddle Point iterations. Active Set returns markedly better bounds than Big-M in the same time, demonstrating the benefit of operating (at least partly) on the tighter dual (5). On the other hand, Saddle Point is rarely beneficial with respect to Big-M when running it for a few iterations.

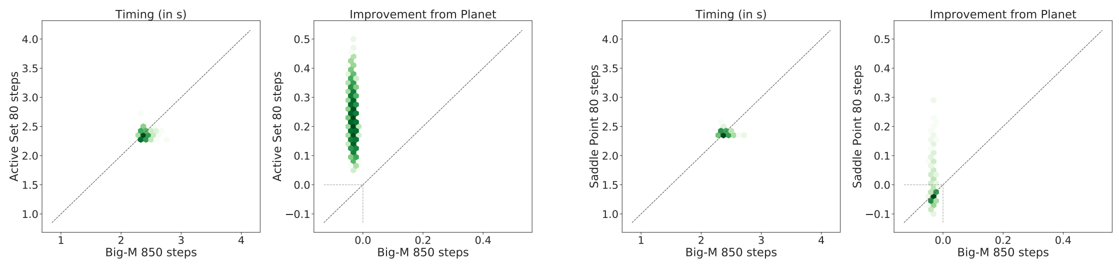
Figure 3 compares the performance of Active Set and Saddle Point for different runtimes, once again on the data from Figure 1. While Active Set yields better average bounds when fewer iterations are employed, the performance gap shrinks with increasing computational budgets, and AS-SP (Active Set initialization to Saddle Point) yields tighter average bounds than Active Set in the same time. Differently from Active Set, the memory footprint of Saddle Point does not increase with the number of iterations (see §5). Therefore, we believe the Frank-Wolfe-based algorithm is particularly convenient in incomplete verification settings that require tight bounds.

8.2.2 Memory Efficiency

As stated in §5, one of the main advantages of Saddle Point is its memory efficiency. In fact, differently from Active Set, the attainable bounding tightness will not depend on the available memory. In order to illustrate this, we present results on a large fully connected network with two hidden layers of width 7000. We trained the network adversarially (Madry



(a) Comparison of runtime (left) and gap to Gurobi Planet bounds (right). For the latter, lower is better.



(b) Comparison of runtime (Timing) and difference with the Gurobi Planet bounds (Improvement from Planet). For the latter, higher is better.

Figure 2: Pointwise comparison for a subset of the methods on the data presented in Figure 1. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

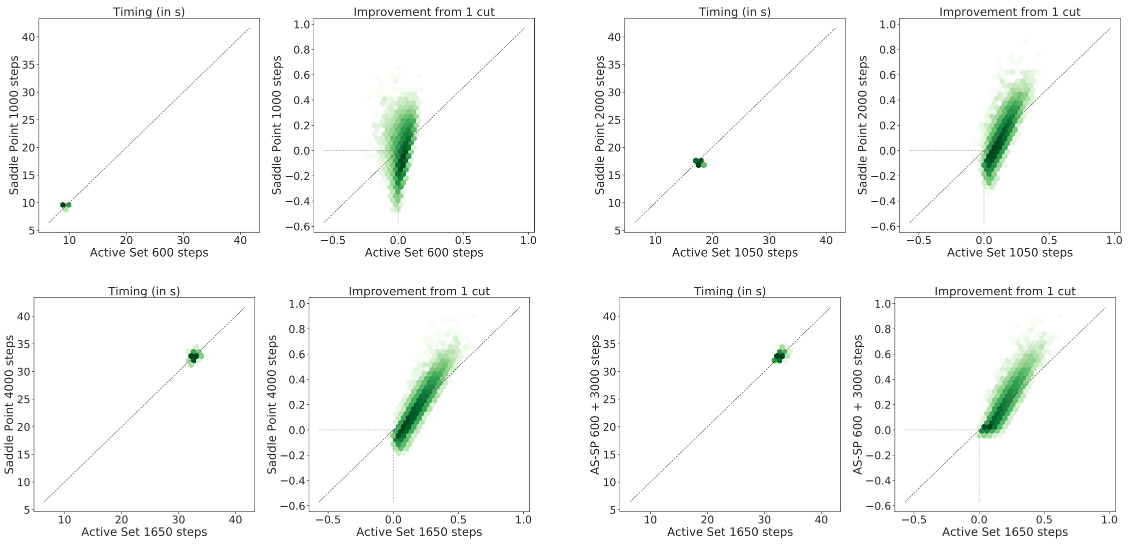


Figure 3: Pointwise comparison between our proposed solvers on the data presented in Figure 1. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

et al., 2018) against perturbations of size $\epsilon_{train} = 2/255$, which is the same radius that we employ at verification time. On the Nvidia Titan Xp GPU employed for our experiments, Active Set is only able to include a single constraint from $\mathcal{A}_{\mathcal{E},k}$ per neuron without running out of memory. Except the maximum allowed number of cuts for Active Set, we run all algorithms with the same hyper-parameters as in §8.2.1. We conduct the experiment on the first 500 images of the CIFAR-10 test set. Figure 4 shows that, while Active Set is competitive with Saddle Point when both are run for a few iterations, Saddle Point yields significantly better speed-accuracy trade-offs when both algorithms are run for longer. Indeed, the use of a single tightening constraint per neuron severely limits the tightness attainable by Active Set, which yields looser bounds than Gurobi cut on average. This is dissimilar from Figure 1, where Active Set rapidly overcomes Gurobi cut in terms of bounding tightness. On the other hand, Saddle Point returns bounds that are markedly tighter than those from the primal baselines in a fraction of their runtime, highlighting its benefits in memory-intensive settings.

8.3 Branch and Bound

We now assess the effectiveness of our algorithms within branch and bound (see §6.2). In particular, we will employ them within BaBSR (Bunel et al., 2020b). In BaBSR, branching is carried out by splitting an unfixed ReLU into its passing and blocking phases (see §6.2 for a description of branch and bound). In order to determine which ReLU to split on, BaBSR employs an inexpensive heuristic based on the bounding algorithm by Wong and Kolter (2018). The goal of the heuristic is to assess which ReLU induces maximum change in the domain's lower bound when made unambiguous.

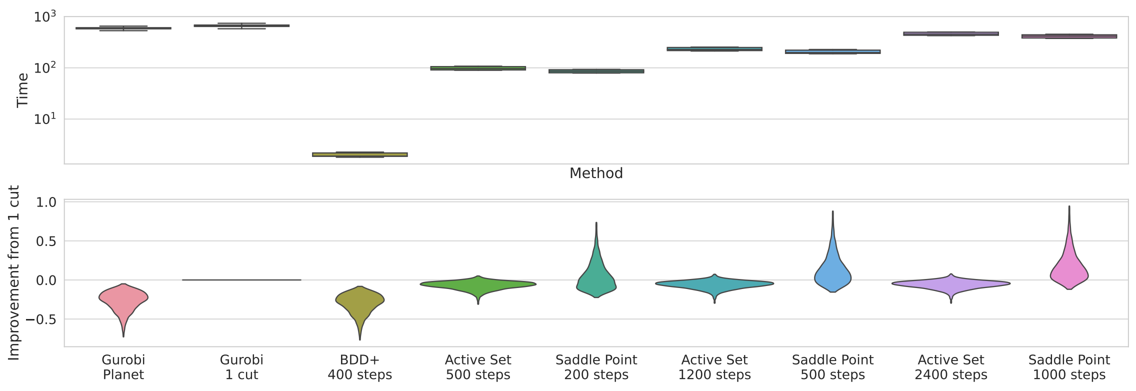


Figure 4: Upper bounds to the adversarial vulnerability of a fully connected network with two hidden layers of width 7000. Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. When run for enough iterations, Saddle Point achieves bounds tighter than both Gurobi 1 cut and Active Set, whose tightness is constrained by memory, in less time.

8.3.1 Complete Verification

We evaluate the performance on complete verification by verifying the adversarial robustness of a network to perturbations in ℓ_∞ norm on a subset of the data set by Lu and Kumar (2020). We replicate the experimental setting from Bunel et al. (2020a).

Lu and Kumar (2020) provide, for a subset of the CIFAR-10 test set, a verification radius ϵ_{ver} defining the small region over which to look for adversarial examples (input points for which the output of the network is not the correct class) and a (randomly sampled) non-correct class to verify against. The verification problem is formulated as the search for an adversarial example, carried out by minimizing the difference between the ground truth logit and the target logit. If the minimum is positive, we have not succeeded in finding a counter-example, and the network is robust. The ϵ_{ver} radius was tuned to meet a certain "problem difficulty" via binary search, employing a Gurobi-based bounding algorithm (Lu and Kumar, 2020). In particular, Lu and Kumar (2020) chose perturbation radii to rule out properties for which an adversarial example can be rapidly found, and properties for which Gurobi Planet would be able to prove robustness without any branching. This characteristic makes the data set an appropriate testing ground for tighter relaxations like the one by Anderson et al. (2020) (§2.2). The networks are robust on all the properties we employed. Three different network architectures of different sizes are used, all robustly trained for $\epsilon_{train} = 2/255$ with the algorithm by Wong and Kolter (2018). A "Base" network with 3172 ReLU activations, and two networks with roughly twice as many activations: one "Deep", the other "Wide". Details can be found in Table 1. We restricted the original data set to 100 properties per network so as to mirror the setup of the recent VNN-COMP competition (VNN-COMP, 2020). The properties have an average perturbation radius of $\epsilon_{ver} = {}^{10.1}/_{255}$, $\epsilon_{ver} = {}^{6.9}/_{255}$, $\epsilon_{ver} = {}^{7.1}/_{255}$ for the Base, Wide, and Deep networks, respec-

Network Name	No. of Properties	Network Architecture
BASE Model	100	Conv2d(3,8,4, stride=2, padding=1) Conv2d(8,16,4, stride=2, padding=1) linear layer of 100 hidden units linear layer of 10 hidden units (Total ReLU activation units: 3172)
WIDE	100	Conv2d(3,16,4, stride=2, padding=1) Conv2d(16,32,4, stride=2, padding=1) linear layer of 100 hidden units linear layer of 10 hidden units (Total ReLU activation units: 6244)

DEEP	100	Conv2d(3,8,4, stride=2, padding=1)
		Conv2d(8,8,3, stride=1, padding=1)
		Conv2d(8,8,3, stride=1, padding=1)
		Conv2d(8,8,4, stride=2, padding=1)
		linear layer of 100 hidden units
		linear layer of 10 hidden units
		(Total ReLU activation units: 6756)

Table 1: For each complete verification experiment, the network architecture used and the number of verification properties tested, a subset of the data set by Lu and Kumar (2020). Each layer but the last is followed by a ReLU activation function.

tively.

We compare the effect on final verification time of using the different bounding methods in §8.2 within BaBSR. When stratifying two bounding algorithms (see §6.2) we denote the resulting method by the names of both the looser and the tighter bounding method, separated by a plus sign (for instance, **Big-M + Active Set**). In addition, we compare against the following complete verification algorithms:

- **MIP** $\mathcal{A}_k$ encodes problem (1) as a Big-M MIP (Tjeng et al., 2019) and solves it in Gurobi by adding cutting planes from $\mathcal{A}_k$. This mirrors the original experiments from Anderson et al. (2020).
- **ERAN** (Singh et al., 2020), a state-of-the-art complete verification toolbox. Results on the data set by Lu and Kumar (2020) are taken from the recent VNN-COMP competition[4] (VNN-COMP, 2020).
- **Fast-and-Complete** (Xu et al., 2021): a recent complete verifier based on BaBSR that pairs fast dual bounds with Gurobi Planet to obtain state-of-the-art performance.

We use 100 iterations for BDD+ (as done by Bunel et al. (2020a)) and 180 for Big-M, which was tuned to employ roughly the same time per bounding computation as BDD+. We re-employed the same hyper-parameters for Big-M, Active Set and Saddle Point, except

4. These were executed by Singh et al. (2020) on a 2.6 GHz Intel Xeon CPU E5-2690 with 512 GB of main memory, utilising 14 cores.

	Base			Wide			Deep		
Method	time(s)	sub-problems	%Timeout	time(s)	sub-problems	%Timeout	time(s)	sub-problems	%Timeout
BDD+ BaBSR	883.55	82 699.40	22.00	568.25	43 751.88	13.00	281.47	10 763.48	5.00
Big-M BaBSR	826.60	68 582.00	19.00	533.79	35 877.24	12.00	253.37	9346.78	4.00
A. Set 100 it. BaBSR	422.32	**9471.90**	**7.00**	169.73	**1873.36**	**3.00**	227.26	2302.16	2.00
Big-M + A. Set 100 it. BaBSR	**415.20**	10 449.10	**7.00**	**163.02**	2402.28	**3.00**	199.70	2709.60	2.00
G. Planet + G. 1 cut BaBSR	949.06	1572.10	15.00	762.42	514.02	6.00	799.71	391.70	2.00
MIP $\mathcal{A}_k$	3227.50	226.24	82.00	2500.70	100.93	64.00	3339.37	434.57	91.00
ERAN	805.89	-	5.00	632.12	-	9.00	545.72	-	0.00
Fast-and-Complete	711.63	9801.22	16.00	350.57	8699.74	8.00	**56.52**	**2238.52**	**1.00**

Table 2: We compare average solving time, average number of solved sub-problems and the percentage of timed out properties on data from Lu and Kumar (2020). The best dual iterative method is highlighted in bold.

the number of iterations. For dual iterative algorithms, we solve 300 subproblems at once for the base network and 200 for the deep and wide networks (see §6.1). Additionally, dual variables are initialized from their parent node's bounding computation. As in Bunel et al. (2020a), the time-limit is kept at one hour.

Figure 2(b) in the previous section shows that Active Set can yield a relatively large improvement over Gurobi Planet bounds, the convex barrier as defined by Salman et al. (2019), in a few iterations. In light of this, we start our experimental evaluation by comparing the performance of 100 iterations of Active Set (within BaBSR) with the relevant baselines. Figure 5 and Table 2 show that Big-M performs competitively with BDD+. With respect to BDD+ and Big-M, which operate on the looser formulation (2), Active Set verifies a larger share of properties and yields faster average verification. This demonstrates the benefit of tighter bounds (§8.2) in complete verification. On the other hand, the poor performance of MIP + $\mathcal{A}_k$ and of Gurobi Planet + Gurobi 1 cut, tied to scaling limitations of off-the-shelf solvers, shows that tighter bounds are effective only if they can be computed efficiently. Nevertheless, the difference in performance between the two Gurobi-based methods confirms that customized Branch and Bound solvers (BaBSR) are preferable to generic MIP solvers, as observed by Bunel et al. (2020b) on the looser Planet relaxation. Moreover,

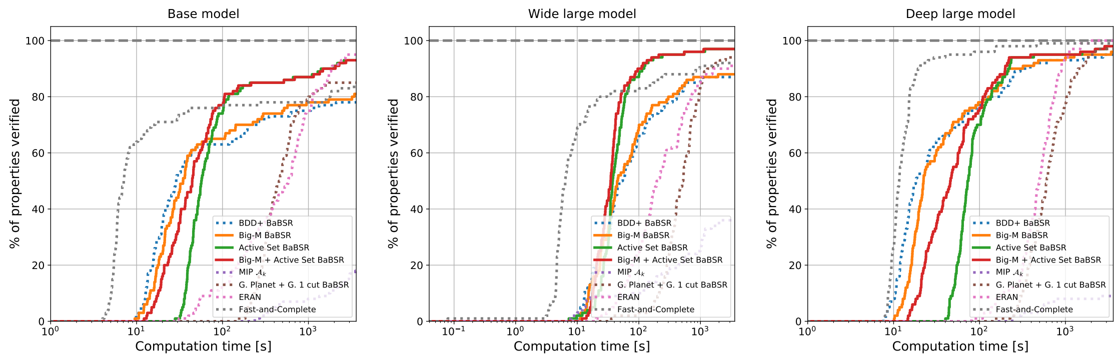


Figure 5: Cactus plots on properties from Lu and Kumar (2020), displaying the percentage of solved properties as a function of runtime. Baselines are represented by dotted lines.

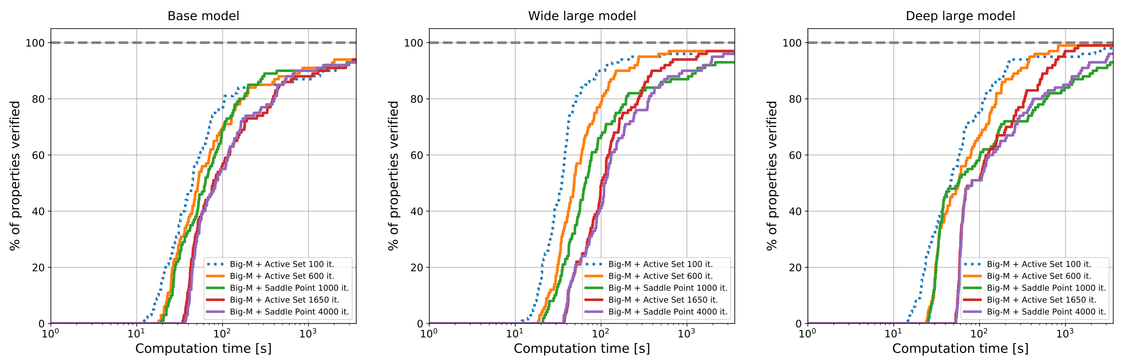


Figure 6: Cactus plots on properties from Lu and Kumar (2020), displaying the percentage of solved properties as a function of runtime. Comparison of best performing method from Figure 5 with tighter bounding schemes.

the stratified bounding system allows us to retain some of the speed of Big-M on easier properties, without sacrificing Active Set's gains on the harder ones. While ERAN verifies 2% more properties than Active Set on two networks, BaBSR (with any dual bounding algorithm) is faster on most of the properties. Fast-and-Complete performs particularly well on the easier properties and on the Deep model, where it is the fastest algorithm. Nevertheless, it struggles on the harder properties, leading to larger average verification times and more timeouts than Active Set on the Base and Wide models. We believe that stratifying Active Set on the inexpensive dual bounds from Fast-and-Complete, which fall short of the convex barrier yet jointly tighten the intermediate bounds, would preserve the advantages of both methods. BaBSR-based results could be further improved by employing the learned branching strategy presented by Lu and Kumar (2020): in this work, we focused on the bounding component of branch and bound.

The results in Figure 5 demonstrate that a small yet inexpensive tightening of the Planet bounds yields large complete verification improvements. We now investigate the effect of employing even tighter, yet more costly, bounds from our solvers. To this end, we compare 100 iterations of Active Set with more expensive bounding schemes from our incomplete verification experiments (§8.2). All methods were stratified along Big-M to

	Base			Wide			Deep		
Method	time(s)	sub-problems	%Timeout	time(s)	sub-problems	%Timeout	time(s)	sub-problems	%Timeout
Big-M + Active Set 100 it.	415.20	10 449.10	7.00	**163.02**	2402.28	**3.00**	199.70	2709.60	2.00
Big-M + Active Set 600 it.	**360.16**	**4806.14**	**6.00**	181.27	**1403.90**	**3.00**	**148.06**	**1061.90**	**1.00**
Big-M + Saddle Point 1000 it.	382.15	5673.04	7.00	417.41	1900.90	7.00	540.68	2551.62	7.00
Big-M + Active Set 1650 it.	463.00	7484.54	**6.00**	285.99	1634.98	**3.00**	250.52	1119.00	**1.00**
Big-M + Saddle Point 4000 it.	434.92	4859.68	7.00	402.86	1703.48	**3.00**	482.93	1444.20	4.00

Table 3: We compare average solving time, average number of solved sub-problems and the percentage of timed out properties on data from Lu and Kumar (2020). The best result is highlighted in bold. Comparison of best performing method from Figure 5 with tighter bounding schemes within BaBSR.

improve performance on easier properties (see §6.2) and employed within BaBSR. Figure 6 and Table 3 show results for two different bounding budgets: 600 iterations of Active Set or 1000 of Saddle Point, 1650 iterations of Active Set or 4000 of Saddle Point (see Figure 3). Despite their ability to prune more subproblems, due to their large computational cost, tighter bounds do not necessarily correspond to shorter overall verification times. Running Active Set for 600 iterations of Active Set leads to faster verification of the harder properties while slowing it down for the easier ones. On the base and deep model, the benefits lead to a smaller average runtime (Table 3). This does not happen on the wide network, for which not enough subproblems are pruned. On the other hand, 1650 Active Set iterations do not prune enough subproblems for their computational overhead, leading to slower formal verification. The behavior of Saddle Point mirrors what was seen for incomplete verification: while it does not perform as well as Active Set for small computational budgets, the gap shrinks when the algorithms are run for more iterations and it is very competitive with Active Set on the base network. Keeping in mind that the memory cost of each variable addition to the active set is larger in complete verification due to subproblem batching (see 6.1), Saddle Point constitutes a valid complete verification alternative in settings where memory is critical.

8.3.2 Branch and Bound for Incomplete Verification

When run for a fixed number of iterations, branch and bound frameworks can be effectively employed for incomplete verification. As shown in §8.3.1, one of the main advantages of our algorithms is their integration and performance within branch and bound. We provide further evidence of this by comparing them with previous algorithms that overcame the convex barrier, but which were not designed for employment within branch and bound: **kPoly** by Singh et al. (2019a), **Fast2CV** and **Opt2CV** by Tjandraatmadja et al. (2020). In particular, we compute the number of verified images, against perturbations of radius $\epsilon_{ver} = 2/255$, on the first 1000 examples of the CIFAR-10 test set for the adversarially-trained (Madry et al., 2018) `ConvSmall` network from the ERAN (Singh et al., 2020) data set, excluding misclassified images. We test different speed-accuracy trade-offs within branch and bound: BDD+ is run for 400 iterations per branch and bound sub-problem, and at most 5 branch and bound batches. Active Set and Saddle Point are run for 200 and 4000 iterations, respectively, with at most 10 batches within BaBSR. Hyper-parameters are kept as in §8.2. This experiment is run on a single Nvidia Titan V GPU. Table 4 shows that, regardless of the employed speed-accuracy trade-off, the use of branch and bound is beneficial with respect to kPoly, Fast2CV, and Opt2CV. Therefore, the use of relaxations

	kPoly	FastC2V	OptC2V	BDD+ BaBSR	Active Set BaBSR	Saddle Point BaBSR
Verified Properties	399	390	398	401	434	408
Average Runtime [s]	86	15.3	104.8	2.5	7.0	43.16

Table 4: Number of verified properties and average runtime on the adversarially trained (Madry et al., 2018) `ConvSmall` network from the ERAN (Singh et al., 2020) data set, on the first 1000 images of the CIFAR-10 test set. Results for FastC2V and OptC2V are taken from Tjandraatmadja et al. (2020), results for kPoly are taken from Singh et al. (2019a).

tighter than Planet does not necessarily improve incomplete verification performance. Increasing the computational budget of branch and bound results in a larger number of verified properties, as demonstrated by the performance of Active Set and Saddle Point.

9. Discussion

The vast majority of neural network bounding algorithms focuses on (solving or loosening) a popular triangle-shaped relaxation, referred to as the "convex barrier" for verification. Relaxations that are tighter than this convex barrier have been recently introduced, but the complexity of the standard solvers for such relaxations hinders their applicability. We have presented two different sparse dual solvers for one such relaxation, and empirically demonstrated that they can yield significant formal verification speed-ups. Our results show that tightness, when paired with scalability, is key to the efficiency of neural network verification and instrumental in the definition of a more appropriate "convex barrier". We believe that new customized solvers for similarly tight relaxations are a crucial avenue for future research in the area, possibly beyond piecewise-linear networks. Finally, as it is inevitable that tighter bounds will come at a larger computational cost, future verification systems will be required to recognise a priori whether tight bounds are needed for a given property. A possible solution to this problem could rely on learning algorithms.

Acknowledgments

ADP was supported by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems, grant EP/L015987/1, and an IBM PhD fellowship. HSB was supported using a Tencent studentship through the University of Oxford.

Appendix A. Limitations of Previous Dual Approaches

In this section, we show that previous dual derivations (Bunel et al., 2020a; Dvijotham et al., 2018) violate Fact 1. Therefore, they are not efficiently applicable to problem (3), motivating our own derivation in Section 3.

We start from the approach by Dvijotham et al. (2018), which relies on relaxing equality constraints (1b), (1c) from the original non-convex problem (1). Dvijotham et al. (2018) prove that this relaxation corresponds to solving convex problem (2), which is equivalent to the Planet relaxation (Ehlers, 2017), to which the original proof refers. As we would like to solve tighter problem (3), the derivation is not directly applicable. Relying on intuition from convex analysis applied to duality gaps (Lemaréchal, 2001), we conjecture that relaxing the composition (1c) $\circ$ (1b) might tighten the primal problem equivalent to the relaxation, obtaining the following dual:

$$\begin{aligned} \max_{\boldsymbol{\mu}} \min_{\mathbf{x}} \quad & W_n \mathbf{x}_{n-1} + \mathbf{b}_n + \sum_{k=1}^{n-1} \boldsymbol{\mu}_k^T \left(\mathbf{x}_k - \max\left\{ W_k \mathbf{x}_{k-1} + \mathbf{b}_k, 0 \right\} \right) \\ \text{s.t.} \quad & \mathbf{l}_k \leq \mathbf{x}_k \leq \mathbf{u}_k \qquad\qquad k \in [\![1, n-1]\!], \\ & \mathbf{x}_0 \in \mathcal{C}. \end{aligned} \tag{16}$$

Unfortunately dual (16) requires an LP (the inner minimisation over $\mathbf{x}$, which in this case does not decompose over layers) to be solved exactly to obtain a supergradient and any time a valid bound is needed. This is markedly different from the original dual by Dvijotham et al. (2018), which had an efficient closed-form for the inner problems.

The derivation by Bunel et al. (2020a), instead, operates by substituting (1c) with its convex hull and solving its Lagrangian Decomposition dual. The Decomposition dual for the convex hull of (1c) $\circ$ (1b) (i.e., $\mathcal{A}_k$) takes the following form:

$$\begin{aligned} \max_{\boldsymbol{\rho}} \min_{\mathbf{x},\mathbf{z}} \; & W_n \mathbf{x}_{A,n-1} + \mathbf{b}_n + \sum_{k=1}^{n-1} \boldsymbol{\rho}_k^T \left(\mathbf{x}_{B,k} - \mathbf{x}_{A,k} \right) \\ \text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, \\ & (\mathbf{x}_{B,k}, W_n \mathbf{x}_{A,n-1} + \mathbf{b}_n, \mathbf{z}_k) \in \mathcal{A}_{\text{dec},k} \qquad k \in [\![1, n-1]\!], \end{aligned} \tag{17}$$

where $\mathcal{A}_{\text{dec},k}$ corresponds to $\mathcal{A}_k$ with the following substitutions: $\mathbf{x}_k \rightarrow \mathbf{x}_{B,k}$, and $\hat{\mathbf{x}}_k \rightarrow W_n \mathbf{x}_{A,n-1} + \mathbf{b}_n$. It can be easily seen that the inner problems (the inner minimisation over $\mathbf{x}_{A,k}, \mathbf{x}_{B,k}$, for each layer $k > 0$) are an exponentially sized LP. Again, this differs from the original dual on the Planet relaxation (Bunel et al., 2020a), which had an efficient closed-form for the inner problems.

Appendix B. Dual Initialisation

Algorithm 3 Big-M solver

```
1: function BIGM_COMPUTE_BOUNDS($\{W_k, \mathbf{b}_k, \mathbf{l}_k, \mathbf{u}_k, \hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k\}_{k=1..n}$)
2:     Initialize duals $\boldsymbol{\alpha}^0, \boldsymbol{\beta}^0_{\mathcal{M}}$ using interval propagation bounds (Gowal et al., 2018)
3:     for $t \in [\![1, T-1]\!]$ do
4:         $\mathbf{x}^{*,t}, \mathbf{z}^{*,t} \in \operatorname{argmin}_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^t, \boldsymbol{\beta}^t_{\mathcal{M}})$ using (20)-(21)
5:         $\boldsymbol{\alpha}^{t+1}, \boldsymbol{\beta}^{t+1}_{\mathcal{M}} \leftarrow [\boldsymbol{\alpha}^t, \boldsymbol{\beta}^t] + H[\nabla_{\boldsymbol{\alpha}} d_{\mathcal{M}}(\boldsymbol{\alpha}, \boldsymbol{\beta}), \nabla_{\boldsymbol{\beta}_{\mathcal{M}}} d_{\mathcal{M}}(\boldsymbol{\alpha}, \boldsymbol{\beta})]$    ▷ supergradient step, using (22)
6:         $\boldsymbol{\alpha}^{t+1}, \boldsymbol{\beta}^{t+1}_{\mathcal{M}} \leftarrow \max(\boldsymbol{\alpha}^{t+1}, 0), \max(\boldsymbol{\beta}^{t+1}_{\mathcal{M}}, 0)$    ▷ dual projection
7:     end for
8:     return $\min_{\mathbf{x},\mathbf{z}} \mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}^T, \boldsymbol{\beta}^T_{\mathcal{M}})$
9: end function
```

As shown in Section 3, the Active Set solver reduces to a dual solver for the Big-M relaxation (2) if the active set $\mathcal{B}$ is kept empty throughout execution. We employ this Big-M solver as dual initialization for both Active Set (§4) and Saddle Point (§5). We demonstrate experimentally in §8 that, when used as a stand-alone solver, our Big-M solver is competitive with previous dual algorithms for problem (2).

The goal of this section is to explicitly describe the Big-M solver, which is summarised in algorithm 3. We point out that, in the notation of restricted variable sets from Section 4, $\boldsymbol{\beta}_{\mathcal{M}} := \boldsymbol{\beta}_{\emptyset}$. We now describe the equivalence between the Big-M and Planet relaxations, before presenting the solver in Section B.3 and the dual it operates on in Section B.2.

B.1 Equivalence to Planet

As previously shown (Bunel et al., 2018), the Big-M relaxation ($\mathcal{M}_k$, when considering the k-th layer only) in problem (2) is equivalent to the Planet relaxation by Ehlers (2017). Then, due to strong duality, our Big-M solver (Section B.2) and the solvers by Bunel et al. (2020a); Dvijotham et al. (2018) will all converge to the bounds from the solution of problem (2). In fact, the Decomposition-based method (Bunel et al., 2020a) directly operates on the Planet relaxation, while Dvijotham et al. (2018) prove that their dual is equivalent to doing so.

On the k-th layer, the Planet relaxation takes the following form:

$$\mathcal{P}_k := \begin{cases} \text{if } \hat{\mathbf{l}}_k \leq 0 \text{ and } \hat{\mathbf{u}}_k \geq 0 : & \\ & \mathbf{x}_k \geq 0, \quad \mathbf{x}_k \geq \hat{\mathbf{x}}_k, \\ & \mathbf{x}_k \leq \hat{\mathbf{u}}_\mathbf{k} \odot (\hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k) \odot \left(1/(\hat{\mathbf{u}}_\mathbf{k} - \hat{\mathbf{l}}_\mathbf{k})\right). \\ \text{if } \hat{\mathbf{u}}_k \leq 0 : & \\ & \mathbf{x}_k = 0. \\ \text{if } \hat{\mathbf{l}}_k \geq 0 : & \\ & \mathbf{x}_k = \hat{\mathbf{x}}_k. \end{cases} \tag{18}$$

It can be seen that $\mathcal{P}_k = \text{Proj}_{\mathbf{x},\hat{\mathbf{x}}}\left(\mathcal{M}_k\right)$, where $\text{Proj}_{\mathbf{x},\hat{\mathbf{x}}}$ denotes projection on the $\mathbf{x}, \hat{\mathbf{x}}$ hyperplane. In fact, as $\mathbf{z}_k$ does not appear in the objective of the primal formulation (2), but only in the constraints, this means assigning it the value that allows the largest possible feasible region. This is trivial for passing or blocking ReLUs. For the ambiguous case, instead, Figure 7 (on a single ReLU) shows that $z_k = \frac{\hat{x}_k - \hat{l}_k}{\hat{u}_k - \hat{l}_k}$ is the correct assignment.

B.2 Big-M Dual

As evident from problem (3), $\mathcal{A}_k \subseteq \mathcal{M}_k$. If we relax all constraints in $\mathcal{M}_k$ (except, again, the box constraints), we are going to obtain a dual with a strict subset of the variables in problem (5). The Big-M dual is a specific instance of the Active Set dual (7) where $\mathcal{B} = \emptyset$, and it takes the following form:

$$\begin{aligned}
&\max_{(\boldsymbol{\alpha},\boldsymbol{\beta})\geq 0} d_{\mathcal{M}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}}) \qquad \text{where:} \qquad d_{\mathcal{M}}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}}) := \min_{\mathbf{x},\mathbf{z}}\ \mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}}),\\
&\mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}}) = \left[\begin{array}{l}
-\sum_{k=0}^{n-1}\left(\boldsymbol{\alpha}_k - W_{k+1}^T\boldsymbol{\alpha}_{k+1} - (\boldsymbol{\beta}_{k,0} + \boldsymbol{\beta}_{k,1} - W_{k+1}^T\boldsymbol{\beta}_{k+1,1})\right)^T \mathbf{x}_k\\
+\sum_{k=1}^{n-1}\mathbf{b}_k^T\boldsymbol{\alpha}_k - \sum_{k=1}^{n-1}\left(\boldsymbol{\beta}_{k,0}\odot\hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1}\odot\hat{\mathbf{l}}_k\right)^T \mathbf{z}_k\\
+\sum_{k=1}^{n-1}(\hat{\mathbf{l}}_k - \mathbf{b}_k)^T\boldsymbol{\beta}_{k,1}
\end{array}\right.\\
&\text{s.t.} \qquad \mathbf{x}_0 \in \mathcal{C}, \qquad (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] \qquad k \in [\![1, n-1]\!].
\end{aligned} \tag{19}$$

B.3 Big-M Solver

We initialise dual variables from interval propagation bounds (Gowal et al., 2018): this can be easily done by setting all dual variables except $\boldsymbol{\alpha}_n$ to 0. Then, we can maximize $d_{\mathcal{M}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ via projected supergradient ascent, exactly as described in Section 4 on a generic active set $\mathcal{B}$. All the computations in the solver follow from keeping $\mathcal{B} = \emptyset$ in §4. We explicitly report them here for the reader's convenience.

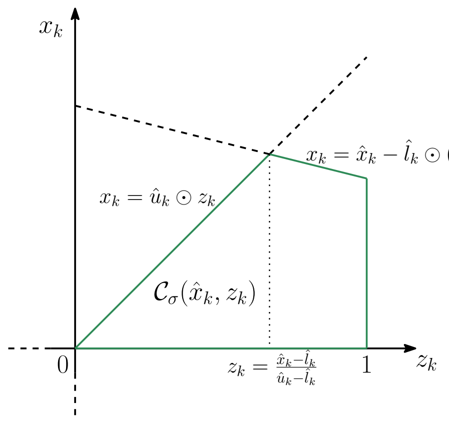

x_k
$x_k = \hat{x}_k - \hat{l}_k \odot (1 - z_k)$
$x_k = \hat{u}_k \odot z_k$
$\mathcal{C}_\sigma(\hat{x}_k, z_k)$
0
$z_k = \frac{\hat{x}_k - \hat{l}_k}{\hat{u}_k - \hat{l}_k}$
1
z_k

Figure 7: $\mathcal{M}_k$ plotted on the $(\mathbf{z}_k, \mathbf{x}_k)$ plane, under the assumption that $\hat{\mathbf{l}}_k \leq 0$ and $\hat{\mathbf{u}}_k \geq 0$.

Let us define the following shorthand for the primal coefficients:

$$\boldsymbol{f}_{\mathcal{M},k}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}}) = \left(\boldsymbol{\alpha}_k - W_{k+1}^T \boldsymbol{\alpha}_{k+1} - (\boldsymbol{\beta}_{k,0} + \boldsymbol{\beta}_{k,1} - W_{k+1}^T \boldsymbol{\beta}_{k,1})\right)$$
$$\boldsymbol{g}_{\mathcal{M},k}(\boldsymbol{\beta}_{\mathcal{M}}) = \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k.$$

The minimisation of the Lagrangian $\mathcal{L}_{\mathcal{M}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ over the primals for $k \in [\![1, n-1]\!]$ is as follows:

$$\mathbf{x}_k^* = \mathbb{1}_{\boldsymbol{f}_{\mathcal{M},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{M}}) \geq 0} \odot \hat{\mathbf{u}}_k + \mathbb{1}_{\boldsymbol{f}_{\mathcal{M},k}(\boldsymbol{\alpha},\boldsymbol{\beta}_{\mathcal{M}}) < 0} \odot \hat{\mathbf{l}}_k \qquad \mathbf{z}_k^* = \mathbb{1}_{\boldsymbol{g}_{\mathcal{M},k}(\boldsymbol{\beta}_{\mathcal{M}}) \geq 0} \odot \mathbf{1} \tag{20}$$

For $k = 0$, instead (assuming, as §4 that this can be done efficiently):

$$\mathbf{x}_0^* \in \underset{\mathbf{x}_0}{\operatorname{argmin}} \quad \boldsymbol{f}_{\mathcal{M},k}(\boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{M}})^T \mathbf{x}_0 \qquad \text{s.t.} \quad \mathbf{x}_0 \in \mathcal{C}. \tag{21}$$

The supergradient over the Big-M dual variables $\boldsymbol{\alpha}, \boldsymbol{\beta}_{k,0}, \boldsymbol{\beta}_{k,1}$ is computed exactly as in §4 and is again a subset of the supergradient of the full dual problem (5). We report it for completeness. For each $k \in [\![0, n-1]\!]$:

$$\begin{gathered}\nabla_{\boldsymbol{\alpha}_k} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = W_k \mathbf{x}_{k-1}^* + \mathbf{b}_k - \mathbf{x}_k^*, \quad \nabla_{\boldsymbol{\beta}_{k,0}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{x}_k - \mathbf{z}_k \odot \hat{\mathbf{u}}_k, \\ \nabla_{\boldsymbol{\beta}_{k,1}} d(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{x}_k - (W_k \mathbf{x}_{k-1} + \mathbf{b}_k) + (1 - \mathbf{z}_k) \odot \hat{\mathbf{l}}_k.\end{gathered} \tag{22}$$

Appendix C. Implementation Details for Saddle Point

In this section, we present details of the Saddle Point solver that were omitted from the main paper. We start with the choice of price caps $\boldsymbol{\mu}_k$ on the Lagrangian multipliers (§C.1), and conclude with a description of our primal initialisation procedure (§C.2).

C.1 Constraint Price Caps

Price caps $\boldsymbol{\mu}$ containing the optimal dual solution to problem (5) can be found via binary search (see §5.1). However, such a strategy might require running Saddle Point to convergence on several instances of problem (13). Therefore, in practice, we employ a heuristic to set constraint caps to a reasonable approximation.

Given duals $(\boldsymbol{\alpha}^0, \boldsymbol{\beta}^0)$ from the dual initialization procedure (algorithm 1 or algorithm 3, depending on the computational budget), for each $k \in [\![1, n-1]\!]$ we set $\boldsymbol{\mu}_k$ as follows:

$$\begin{aligned} \boldsymbol{\mu}_{\alpha,k} &= \begin{cases} \boldsymbol{\alpha}_k^0 & \text{if } \boldsymbol{\alpha}_k^0 > 0 \\ c_{\alpha,k} & \text{otherwise} \end{cases} \\ \boldsymbol{\mu}_{\beta,k} &= \begin{cases} \sum_{I_k \in 2^{W_k}} \boldsymbol{\beta}_k^0 & \text{if } \sum_{I_k \in 2^{W_k}} \boldsymbol{\beta}_k^0 > 0 \\ c_{\beta,k} & \text{otherwise,} \end{cases} \end{aligned} \tag{23}$$

where $c_{\alpha,k}$ and $c_{\beta,k}$ are small positive constants. In other words, we cap the (sums over) dual variables to their values at initialization if these are non-zero, and we allow dual variables to turn positive otherwise. While a larger feasible region (for instance, setting $\boldsymbol{\mu}_{\alpha,k} = \max_i \boldsymbol{\alpha}_k[i] \odot \mathbf{1}$, $\boldsymbol{\mu}_{\beta,k} = \max_i \sum_{I_k \in 2^{W_k}} \boldsymbol{\beta}_k^0[i] \odot \mathbf{1}$) might yield tighter bounds at convergence, we found assignment (23) to be more effective on the iteration budgets employed in §8.2,§8.3.

C.2 Primal Initialization

If we invert the minimization and maximization, our saddle-point problem (13) can be seen as a non-smooth minimization problem (the minimax theorem holds):

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{z}} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} & \left[\begin{array}{l} \sum_{k=1}^{n-1} \boldsymbol{\alpha}_k^T \left(W_k \mathbf{x}_{k-1} + \mathbf{b}_k - \mathbf{x}_k\right) + \sum_{k=1}^{n-1} \boldsymbol{\beta}_{k,0}^T \left(\mathbf{x}_k - \mathbf{z}_k \odot \hat{\mathbf{u}}_k\right) \\ + \sum_{k=1}^{n-1} \sum_{I_k \in \mathcal{E}_k} \boldsymbol{\beta}_{k,I_k}^T \left(\begin{array}{l} \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k) \\ -\mathbf{b}_k \odot \mathbf{z}_k - \left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k \\ - \left(W_k \odot I_k\right) \mathbf{x}_{k-1} + \mathbf{x}_k \end{array} \right) \\ + \sum_{k=1}^{n-1} \boldsymbol{\beta}_{k,1}^T \left(\mathbf{x}_k - \left(W_k \mathbf{x}_{k-1} + \mathbf{b}_k\right) + (1 - \mathbf{z}_k) \odot \hat{\mathbf{l}}_k\right) + W_n \mathbf{x}_{n-1} + \mathbf{b}_n \end{array} \right. \\
\text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C}, \\
& (\mathbf{x}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\mathbf{0},\ \mathbf{1}] \qquad\qquad k \in [\![1, n-1]\!], \\
& \boldsymbol{\alpha}_k \in [\mathbf{0}, \boldsymbol{\mu}_{k,\alpha}] \\
& \left(\boldsymbol{\beta}_k \geq \mathbf{0},\ \sum_{I_k \in \mathcal{E}_k \cup \{0,1\}} \boldsymbol{\beta}_{k,I_k} \leq \boldsymbol{\mu}_{k,\beta}\right) \qquad\qquad k \in [\![1, n-1]\!].
\end{aligned} \tag{24}
$$

The "primal view"[5] of problem (24) admits an efficient subgradient method, which we employ as primal initialization for the Saddle Point solver.

C.2.1 Technical Details

The inner maximizers $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$, which are required to obtain a valid subgradient, will be given in closed-form by the dual conditional gradients from equations (14), (15) (the objective is bilinear). Then, using the dual functions defined in (6), the subgradient over the linearly-many primal variables can be computed as $\boldsymbol{f}_k(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ for $\mathbf{x}_k$ and $\boldsymbol{g}_k(\boldsymbol{\beta}^*)$ for $\mathbf{z}_k$. After each subgradient step, the primals are projected to the feasible space: this is trivial for $\mathbf{x}_k$ and $\mathbf{z}_k$, which are box constrained, and can be efficiently performed for $\mathcal{C}$ in the common cases of ℓ_∞ or ℓ_2 balls.

C.2.2 Simplified Initialization

Due to the additional cost associated to masked forward-backward passes (see appendix G), the primal initialization procedure can be simplified by restricting the dual variables to the ones associated to the Big-M relaxation (appendix B). This can be done by substituting $\mathcal{E}_k \leftarrow \emptyset$ and $\boldsymbol{\beta}_k \leftarrow \boldsymbol{\beta}_{\emptyset,k}$ (see notation in §4.1) in problem (24). A subgradient method for the resulting problem can then be easily adapted from the description above.

Appendix D. Dual Derivations

We now derive problem (5), the dual of the full relaxation by Anderson et al. (2020) described in equation (3). The Active Set (equation (7)) and Big-M duals (equation (19)) can be obtained by removing $\boldsymbol{\beta}_{k,I_k} \forall\ I_k \in \mathcal{E}_k \setminus \mathcal{B}_k$ and $\boldsymbol{\beta}_{k,I_k} \forall\ I_k \in \mathcal{E}_k$, respectively. We employ the following Lagrangian multipliers:

5. due to the restricted dual domain, problem (24) does not correspond to the original primal problem in (3).

$$
\begin{aligned}
\mathbf{x}_k \geq \hat{\mathbf{x}}_k &\Rightarrow \boldsymbol{\alpha}_k, \\
\mathbf{x}_k \leq \hat{\mathbf{u}}_k \odot \mathbf{z}_k &\Rightarrow \boldsymbol{\beta}_{k,0}, \\
\mathbf{x}_k \leq \hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k \odot (1-\mathbf{z}_k) &\Rightarrow \boldsymbol{\beta}_{k,1}, \\
\mathbf{x}_k \leq \begin{pmatrix} (W_k \odot I_k)\,\mathbf{x}_{k-1} + \mathbf{z}_k \odot \mathbf{b}_k \\ -\left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1-\mathbf{z}_k) \\ +\left(W_k \odot (1-I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k \end{pmatrix} &\Rightarrow \boldsymbol{\beta}_{k,I_k},
\end{aligned}
$$

and obtain, as a Lagrangian (using $\hat{\mathbf{x}}_k = W_k\mathbf{x}_{k-1} + \mathbf{b}_k$):

$$
\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \left[\begin{array}{l}
\sum_{k=1}^{n-1} \boldsymbol{\alpha}_k^T \left(W_k\mathbf{x}_{k-1} + \mathbf{b}_k - \mathbf{x}_k\right) + \sum_{k=1}^{n-1} \boldsymbol{\beta}_{k,0}^T \left(\mathbf{x}_k - \mathbf{z}_k \odot \hat{\mathbf{u}}_k\right) \\
+\sum_{k=1}^{n-1}\sum_{I_k\in\mathcal{E}_k} \boldsymbol{\beta}_{k,I_k}^T \begin{pmatrix} \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1-\mathbf{z}_k) - (W_k \odot I_k)\,\mathbf{x}_{k-1} \\ -\mathbf{b}_k \odot \mathbf{z}_k - \left(W_k \odot (1-I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k + \mathbf{x}_k \end{pmatrix} \\
+\sum_{k=1}^{n-1} \boldsymbol{\beta}_{k,1}^T \left(\mathbf{x}_k - (W_k\mathbf{x}_{k-1} + \mathbf{b}_k) + (1-\mathbf{z}_k) \odot \hat{\mathbf{l}}_k\right) + W_n\mathbf{x}_{n-1} + \mathbf{b}_n
\end{array}\right.
$$

Let us use $\sum_{I_k}$ as shorthand for $\sum_{I_k\in\mathcal{E}_k\cup\{0,1\}}$. If we collect the terms with respect to the primal variables and employ dummy variables $\boldsymbol{\alpha}_0 = 0$, $\boldsymbol{\beta}_0 = 0$, $\boldsymbol{\alpha}_n = I$, $\boldsymbol{\beta}_n = 0$, we obtain:

$$
\mathcal{L}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \left[\begin{array}{l}
-\sum_{k=0}^{n-1} \begin{pmatrix} \boldsymbol{\alpha}_k - W_{k+1}^T\boldsymbol{\alpha}_{k+1} - \sum_{I_k} \boldsymbol{\beta}_{k,I_k} \\ +\sum_{I_{k+1}} (W_{k+1} \odot I_{k+1})^T \boldsymbol{\beta}_{k+1,I_{k+1}} \end{pmatrix}^T \mathbf{x}_k \\
-\sum_{k=1}^{n-1} \begin{pmatrix} \sum_{I_k\in\mathcal{E}_k} \boldsymbol{\beta}_{k,I_k} \odot \mathbf{b}_k + \boldsymbol{\beta}_{k,1} \odot \hat{\mathbf{l}}_k + \boldsymbol{\beta}_{k,0} \odot \hat{\mathbf{u}}_k \\ +\sum_{I_k\in\mathcal{E}_k} \left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond \boldsymbol{\beta}_{k,I_k} \\ +\sum_{I_k\in\mathcal{E}_k} \left(W_k \odot (1-I_k) \odot \check{U}_{k-1}\right) \diamond \boldsymbol{\beta}_{k,I_k} \end{pmatrix}^T \mathbf{z}_k \\
+\sum_{k=1}^{n-1} \mathbf{b}_k^T\boldsymbol{\alpha}_k + \sum_{k=1}^{n-1} \left(\sum_{I_k\in\mathcal{E}_k} (W_k \odot I_k \odot \check{L}_{k-1}) \,\square\, \boldsymbol{\beta}_{k,I_k} + \boldsymbol{\beta}_{k,1}^T (\hat{\mathbf{l}}_k - \mathbf{b}_k)\right)
\end{array}\right.
$$

which corresponds to the form shown in problem (5).

Appendix E. Intermediate Bounds

A crucial quantity in both ReLU relaxations ($\mathcal{M}_k$ and $\mathcal{A}_k$) are intermediate pre-activation bounds $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$. In practice, they are computed by solving a relaxation $\mathcal{C}_k$ (which might be $\mathcal{M}_k$, $\mathcal{A}_k$, or something looser) of (1) over subsets of the network (Bunel et al., 2020a). For $\hat{\mathbf{l}}_i$, this means solving the following problem (separately, for each entry $\hat{\mathbf{l}}_i[j]$):

$$
\begin{aligned}
\min_{\mathbf{x}, \hat{\mathbf{x}}, \mathbf{z}} \quad & \hat{\mathbf{x}}_i[j] \\
\text{s.t.} \quad & \mathbf{x}_0 \in \mathcal{C} \\
& \hat{\mathbf{x}}_{k+1} = W_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1}, \quad k \in \llbracket 0, i-1 \rrbracket, \\
& (\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in \mathcal{C}_k \quad\quad k \in \llbracket 1, i-1 \rrbracket.
\end{aligned}
\tag{25}
$$

As (25) needs to be solved twice for each neuron (lower and upper bounds, changing the sign of the last layer's weights) rather than once as in (3), depending on the computational budget, $\mathcal{C}_k$ might be looser than the relaxation employed for the last layer bounds (in our case, $\mathcal{A}_k$). In all our experiments, we compute intermediate bounds as the tightest bounds between the method by Wong and Kolter (2018) and Interval Propagation (Gowal et al., 2018).

Once pre-activation bounds are available, post-activation bounds can be simply computed as $\mathbf{l}_k = \max(\hat{\mathbf{l}}_k, 0), \mathbf{u}_k = \max(\hat{\mathbf{u}}_k, 0)$.

Appendix F. Pre-activation Bounds in $\mathcal{A}_k$

We now highlight the importance of an explicit treatment of pre-activation bounds in the context of the relaxation by Anderson et al. (2020). In §F.1 we will show through an example that, without a separate pre-activation bounds treatment, $\mathcal{A}_k$ could be looser than the less computationally expensive $\mathcal{M}_k$ relaxation. We then (§F.2) justify our specific pre-activation bounds treatment by extending the original proof by Anderson et al. (2020).

The original formulation by Anderson et al. (2020) is the following:

$$\left.\begin{array}{l} \mathbf{x}_k \geq W_k \mathbf{x}_{k-1} + \mathbf{b}_k \\ \mathbf{x}_k \leq \left(\begin{array}{l} (W_k \odot I_k)\, \mathbf{x}_{k-1} + \mathbf{z}_k \odot \mathbf{b}_k \\ -\left(W_k \odot I_k \odot \check{L}_{k-1}\right) \diamond (1 - \mathbf{z}_k) \\ +\left(W_k \odot (1 - I_k) \odot \check{U}_{k-1}\right) \diamond \mathbf{z}_k \end{array}\right) \quad \forall I_k \in 2^{W_k} \\ (\mathbf{x}_k, \hat{\mathbf{x}}_k, \mathbf{z}_k) \in [\mathbf{l}_k, \mathbf{u}_k] \times [\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k] \times [\mathbf{0},\ \mathbf{1}] \end{array}\right\} = \mathcal{A}'_k. \tag{26}$$

While pre-activation bounds regularly appear as lower and upper bounds to $\hat{\mathbf{x}}_k$, they do not appear in any other constraint. Indeed, the difference with respect to $\mathcal{A}_k$ as defined in equation (3) exclusively lies in the treatment of pre-activation bounds within the exponential family. Set $\mathcal{A}_k$ explicitly employs generic $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ in the constraint set through $\mathbf{x}_k \leq \hat{\mathbf{u}}_k \odot \mathbf{z}_k$, and $\mathbf{x}_k \leq \hat{\mathbf{x}}_k - \hat{\mathbf{l}}_k \odot (1 - \mathbf{z}_k)$ from $\mathcal{M}_k$. On the other hand, $\mathcal{A}'_k$ implicitly sets $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ to the value dictated by interval propagation bounds (Gowal et al., 2018) via the constraints in $I_k = 0$ and $I_k = 1$ from the exponential family. In fact, setting $I_k = 0$ and $I_k = 1$, we obtain the following two constraints:

$$\begin{aligned} &\mathbf{x}_k \leq \hat{\mathbf{x}}_k - M_k^- \odot (1 - \mathbf{z}_k) \\ &\mathbf{x}_k \leq M_k^+ \odot \mathbf{z}_k \\ \text{where:} \quad &M_k^- := \min_{\mathbf{x}_{k-1} \in [\mathbf{l}_{k-1}, \mathbf{u}_{k-1}]} W_k^T \mathbf{x}_{k-1} + \mathbf{b}_k = W_k \odot \check{L}_{k-1} + \mathbf{b}_k \\ &M_k^+ := \max_{\mathbf{x}_{k-1} \in [\mathbf{l}_{k-1}, \mathbf{u}_{k-1}]} W_k^T \mathbf{x}_{k-1} + \mathbf{b}_k = W_k \odot \check{U}_{k-1} + \mathbf{b}_k \end{aligned} \tag{27}$$

which correspond to the upper bounding ReLU constraints in $\mathcal{M}_k$ if we set $\hat{\mathbf{l}}_k \rightarrow M_k^-, \hat{\mathbf{u}}_k \rightarrow M_k^+$. While $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ are (potentially) computed solving an optimisation problem over the entire network (problem 25), the optimisation for M_k^-, M_k^+ involves only the layer before the current. Therefore, the constraints in (27) might be much looser than those in $\mathcal{M}_k$.

In practice, the effect of $\hat{\mathbf{l}}_k[i], \hat{\mathbf{u}}_k[i]$ on the resulting set is so significant that $\mathcal{M}_k$ might yield better bounds than $\mathcal{A}'_k$, even on very small networks. We now provide a simple example.

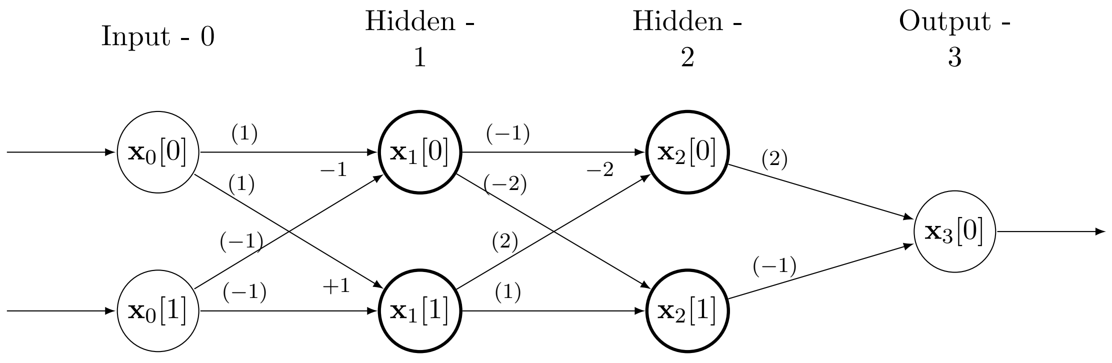


Figure 8: Example network architecture in which $\mathcal{M}_k \subset \mathcal{A}'_k$, with pre-activation bounds computed with $\mathcal{C}_k = \mathcal{M}_k$. For the bold nodes (the two hidden layers) a ReLU activation follows the linear function. The numbers between parentheses indicate multiplicative weights, the others additive biases (if any).

F.1 Motivating Example

Figure 8 illustrates the network architecture. The size of the network is the minimal required to reproduce the phenomenon. $\mathcal{M}_k$ and $\mathcal{A}_k$ coincide for single-neuron layers (Anderson et al., 2020), and $\hat{\mathbf{l}}_k = M_k^-, \hat{\mathbf{u}}_k = M_k^+$ on the first hidden layer (hence, a second layer is needed).

Let us write the example network as a (not yet relaxed, as in problem (1)) optimization problem for the lower bound on the output node $\mathbf{x}_3$.

$$\mathbf{l}_3 = \arg\min_{\mathbf{x},\hat{\mathbf{x}}} \quad \begin{bmatrix} 2 & -1 \end{bmatrix} \mathbf{x}_2 \tag{28a}$$

$$\text{s.t.} \quad \mathbf{x}_0 \in [-1, 1]^2 \tag{28b}$$

$$\hat{\mathbf{x}}_1 = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \mathbf{x}_0 + \begin{bmatrix} -1 \\ 1 \end{bmatrix} \qquad \mathbf{x}_1 = \max(0, \hat{\mathbf{x}}_1) \tag{28c}$$

$$\hat{\mathbf{x}}_2 = \begin{bmatrix} -1 & 2 \\ -2 & 1 \end{bmatrix} \mathbf{x}_1 + \begin{bmatrix} -2 \\ 0 \end{bmatrix} \qquad \mathbf{x}_2 = \max(0, \hat{\mathbf{x}}_2) \tag{28d}$$

$$\mathbf{x}_3 = \begin{bmatrix} 2 & -1 \end{bmatrix} \mathbf{x}_2 \tag{28e}$$

Let us compute pre-activation bounds with $\mathcal{C}_k = \mathcal{M}_k$ (see problem (25)). For this network, the final output lower bound is tighter if the employed relaxation is $\mathcal{M}_k$ rather than $\mathcal{A}_k$ (hence, in this case, $\mathcal{M}_k \subset \mathcal{A}'_k$). Specifically: $\hat{\mathbf{l}}_{3,\mathcal{A}'_k} = -1.2857$, $\hat{\mathbf{l}}_{3,\mathcal{M}_k} = -1.2273$. In fact:

- In order to compute $\mathbf{l}_1$ and $\mathbf{u}_1$, the post-activation bounds of the first-layer, it suffices to solve a box-constrained linear program for $\hat{\mathbf{l}}_1$ and $\hat{\mathbf{u}}_1$, which at this layer coincide with interval propagation bounds, and to clip them to be non-negative. This yields $\mathbf{l}_1 = \begin{bmatrix}0 & 0\end{bmatrix}^T$, $\mathbf{u}_1 = \begin{bmatrix}1 & 3\end{bmatrix}^T$.

- Computing $M_2^+[1] = \max_{\mathbf{x}_1 \in [\mathbf{l}_1, \mathbf{u}_1]} \begin{bmatrix}-2 & 1\end{bmatrix} \mathbf{x}_1 = 3$ we are assuming that $\mathbf{x}_1[0] = \mathbf{l}_1[0]$ and $\mathbf{x}_1[1] = \mathbf{u}_1[1]$. These two assignments are in practice conflicting, as they imply different values for $\mathbf{x}_0$. Specifically, $\mathbf{x}_1[1] = \mathbf{u}_1[1]$ requires $\mathbf{x}_0 = \begin{bmatrix}\mathbf{u}_0[0] & \mathbf{l}_0[1]\end{bmatrix} = \begin{bmatrix}1 & -1\end{bmatrix}$, but this would also imply $\mathbf{x}_1[0] = \mathbf{u}_1[0]$, yielding $\hat{\mathbf{x}}_2[1] = 1 \neq 3$.

  Therefore, explicitly solving a LP relaxation of the network for the value of $\hat{\mathbf{u}}_2[1]$ will tighten the bound. Using $\mathcal{M}_k$, the LP for this intermediate pre-activation bound is:

$$\hat{\mathbf{u}}_2[1] = \arg\min_{\mathbf{x}, \hat{\mathbf{x}}, \mathbf{z}} \quad \begin{bmatrix}-2 & 1\end{bmatrix} \mathbf{x}_1 \tag{29a}$$

$$\text{s.t.} \quad \mathbf{x}_0 \in [-1, 1]^2,\ \mathbf{z}_1 \in [0, 1]^2,\ \mathbf{x}_1 \in \mathbb{R}^2_{\geq 0} \tag{29b}$$

$$\hat{\mathbf{x}}_1 = \begin{bmatrix}1 & -1 \\ 1 & -1\end{bmatrix} \mathbf{x}_0 + \begin{bmatrix}-1 \\ 1\end{bmatrix} \tag{29c}$$

$$\mathbf{x}_1 \geq \hat{\mathbf{x}}_1 \tag{29d}$$

$$\mathbf{x}_1 \leq \hat{\mathbf{u}}_1 \odot \mathbf{z}_1 = \begin{bmatrix}1 \\ 3\end{bmatrix} \odot \mathbf{z}_1 \tag{29e}$$

$$\mathbf{x}_1 \leq \hat{\mathbf{x}}_1 - \hat{\mathbf{l}}_1 \odot (1 - \mathbf{z}_1) = \hat{\mathbf{x}}_1 - \begin{bmatrix}-3 \\ -1\end{bmatrix} \odot (1 - \mathbf{z}_1) \tag{29f}$$

  Yielding $\hat{\mathbf{u}}_2[1] = 2.25 < 3 = M_2^+[1]$. An analogous reasoning holds for $M_2^-[1]$ and $\hat{\mathbf{l}}_2[1]$.

- In $\mathcal{M}_k$, we therefore added the following two constraints:

$$\begin{aligned} \mathbf{x}_2[1] &\leq \hat{\mathbf{x}}_2[1] - \hat{\mathbf{l}}_2[1](1 - \mathbf{z}_2[1]) \\ \mathbf{x}_2[1] &\leq \hat{\mathbf{u}}_2[1]\mathbf{z}_2[1] \end{aligned} \tag{30}$$

  that in $\mathcal{A}'_k$ correspond to the weaker:

$$\begin{aligned} \mathbf{x}_2[1] &\leq \hat{\mathbf{x}}_2[1] - M_2^-[1](1 - \mathbf{z}_2[1]) \\ \mathbf{x}_2[1] &\leq M_2^+[1]\mathbf{z}_2[1] \end{aligned} \tag{31}$$

  As the last layer weight corresponding to $\mathbf{x}_2[1]$ is negative ($W_3[0, 1] = -1$), these constraints are going to influence the computation of $\hat{\mathbf{l}}_3$.

- In fact, the constraints in (30) are both active when optimizing for $\hat{\mathbf{l}}_{3,\mathcal{M}_k}$, whereas their counterparts for $\hat{\mathbf{l}}_{3,\mathcal{A}'_k}$ in (31) are not. The only active upper constraint at neuron $\mathbf{x}_2[1]$ for the Anderson relaxation is $\mathbf{x}_2[1] \leq \mathbf{x}_1[1]$, corresponding to the constraint from $\mathcal{A}'_2$ with $I_2[1, \cdot] = [0\ 1]$. Evidently, its effect is not sufficient to counter-balance the effect of the tighter constraints (30) for $I_2[1, \cdot] = [1\ 1]$ and $I_2[1, \cdot] = [0\ 0]$, yielding a weaker lower bound for the network output.

F.2 Derivation of $\mathcal{A}_k$

Having motivated an explicit pre-activation bounds treatment for the relaxation by Anderson et al. (2020), we now extend the original proof for $\mathcal{A}'_k$ (equation (26)) to obtain our formulation $\mathcal{A}_k$ (as defined in equation (3)). For simplicity, we will operate on a single neuron $\mathbf{x}_k[i]$.

A self-contained way to derive $\mathcal{A}'_k$ is by applying Fourier-Motzkin elimination on a standard MIP formulation referred to as the *multiple choice* formulation (Anderson et al., 2019), which is defined as follows:

$$\left.\begin{array}{l}(\mathbf{x}_{k-1}, \mathbf{x}_k[i]) = (\mathbf{x}^0_{k-1}, \mathbf{x}^0_k[i]) + (\mathbf{x}^1_{k-1}, \mathbf{x}^1_k[i]) \\ \mathbf{x}^0_k[i] = 0 \geq \boldsymbol{w}^T_{i,k}\mathbf{x}^0_{k-1} + \mathbf{b}_k[i](1 - \mathbf{z}_k[i]) \\ \mathbf{x}^1_k[i] = \boldsymbol{w}^T_{i,k}\mathbf{x}^1_{k-1} + \mathbf{b}_k[i]\mathbf{z}_k[i] \geq 0 \\ \mathbf{l}_{k-1}(1 - \mathbf{z}_k[i]) \leq \mathbf{x}^0_{k-1} \leq \mathbf{u}_{k-1}(1 - \mathbf{z}_k[i]) \\ \mathbf{l}_{k-1}\mathbf{z}_k[i] \leq \mathbf{x}^1_{k-1} \leq \mathbf{u}_{k-1}\mathbf{z}_k[i] \\ \mathbf{z}_k[i] \in [0,\ 1]\end{array}\right\} = \mathcal{S}'_{k,i} \tag{32}$$

Where $\boldsymbol{w}_{i,k}$ denotes the i-th row of W_k, and $\mathbf{x}^1_{k-1}$ and $\mathbf{x}^0_{k-1}$ are copies of the previous layer variables. Applying (32) to the entire neural network results in a quadratic number of variables (relative to the number of neurons). The formulation can be obtained from well-known techniques from the MIP literature (Jeroslow, 1987) (it is the union of the two polyhedra for a passing and a blocking ReLU, operating in the space of $\mathbf{x}_{k-1}$). Anderson et al. (2019) show that $\mathcal{A}'_k = \text{Proj}_{\mathbf{x}_{k-1},\mathbf{x}_k,\mathbf{z}_k}(\mathcal{S}'_k)$.

If pre-activation bounds $\hat{\mathbf{l}}_k, \hat{\mathbf{u}}_k$ (computed as described in Section E) are available, we can naturally add them to (32) as follows:

$$\left.\begin{array}{l}(\mathbf{x}_{k-1}, \mathbf{x}_k[i], \mathbf{z}_k[i]) \in \mathcal{S}'_{k,i} \\ \hat{\mathbf{l}}_k[i](1 - \mathbf{z}_k[i]) \leq \boldsymbol{w}^T_{i,k}\mathbf{x}^0_{k-1} + \mathbf{b}_k[i](1 - \mathbf{z}_k[i]) \leq \hat{\mathbf{u}}_k[i](1 - \mathbf{z}_k[i]) \\ \hat{\mathbf{l}}_k[i] \odot \mathbf{z}_k[i] \leq \boldsymbol{w}^T_{i,k}\mathbf{x}^1_{k-1} + \mathbf{b}_k[i]\mathbf{z}_k[i] \leq \hat{\mathbf{u}}_k[i]\mathbf{z}_k[i]\end{array}\right\} = \mathcal{S}_{k,i} \tag{33}$$

We now prove that this formulation yields $\mathcal{A}_k$ when projecting out the copies of the activations.

Proposition 3 *Sets $\mathcal{S}_k$ from equation (33) and $\mathcal{A}_k$ from problem (3) are equivalent, in the sense that $\mathcal{A}_k = Proj_{\mathbf{x}_{k-1},\mathbf{x}_k,\mathbf{z}_k}(\mathcal{S}_k)$.*

Proof In order to prove the equivalence, we will rely on Fourier-Motzkin elimination as in the original Anderson relaxation proof (Anderson et al., 2019). Going along the lines of the original proof, we start from (32) and eliminate $\mathbf{x}^1_{k-1}$, $\mathbf{x}^0_k[i]$ and $\mathbf{x}^1_k[i]$ exploiting the equalities. We then re-write all the inequalities as upper or lower bounds on $\mathbf{x}^0_{k-1}[0]$ in order to eliminate this variable. As Anderson et al. (2019), we assume $\boldsymbol{w}_{i,k}[0] > 0$. The proof generalizes by using $\check{L}$ and $\check{U}$ for $\boldsymbol{w}_{i,k}[0] < 0$, whereas if the coefficient is 0 the variable is easily eliminated. We get the following system:

$$\mathbf{x}^0_{k-1}[0] = \frac{1}{\boldsymbol{w}_{i,k}[0]}\left(\boldsymbol{w}^T_{i,k}\mathbf{x}_{k-1} - \sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i] - \mathbf{x}_k[i]\right) \tag{34a}$$

$$\mathbf{x}^0_{k-1}[0] \leq -\frac{1}{\boldsymbol{w}_{i,k}[0]}\left(\sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i](1-\mathbf{z}_k[i])\right) \tag{34b}$$

$$\mathbf{x}^0_{k-1}[0] \leq \frac{1}{\boldsymbol{w}_{i,k}[0]}\left(\boldsymbol{w}^T_{i,k}\mathbf{x}_{k-1} - \sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i]\right) \tag{34c}$$

$$\mathbf{l}_{k-1}[0](1-\mathbf{z}_k[i]) \leq \mathbf{x}^0_{k-1}[0] \leq \mathbf{u}_{k-1}[0](1-\mathbf{z}_k[i]) \tag{34d}$$

$$\mathbf{x}^0_{k-1}[0] \leq \mathbf{x}_{k-1}[0] - \mathbf{l}_{k-1}[0]\mathbf{z}_k[i] \tag{34e}$$

$$\mathbf{x}^0_{k-1}[0] \geq \mathbf{x}_{k-1}[0] - \mathbf{u}_{k-1}[0]\mathbf{z}_k[i] \tag{34f}$$

$$\mathbf{x}^0_{k-1}[0] \leq \frac{1}{\boldsymbol{w}_{i,k}[0]}\left(\boldsymbol{w}^T_{i,k}\mathbf{x}_{k-1} - \sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + (\boldsymbol{b}_k[i] - \hat{\mathbf{l}}_k[i])\mathbf{z}_k[i]\right) \tag{34g}$$

$$\mathbf{x}^0_{k-1}[0] \geq \frac{1}{\boldsymbol{w}_{i,k}[0]}\left(\boldsymbol{w}^T_{i,k}\mathbf{x}_{k-1} - \sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + (\boldsymbol{b}_k[i] - \hat{\mathbf{u}}_k[i])\mathbf{z}_k[i]\right) \tag{34h}$$

$$\mathbf{x}^0_{k-1}[0] \geq \frac{1}{\boldsymbol{w}_{i,k}[0]}\left((\hat{\mathbf{l}}_k[i] - \boldsymbol{b}_k[i])(1-\mathbf{z}_k[i]) - \sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j]\right) \tag{34i}$$

$$\mathbf{x}^0_{k-1}[0] \leq \frac{1}{\boldsymbol{w}_{i,k}[0]}\left((\hat{\mathbf{u}}_k[i] - \boldsymbol{b}_k[i])(1-\mathbf{z}_k[i]) - \sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j]\right) \tag{34j}$$

where only inequalities (34g) to (34j) are not present in the original proof. We therefore focus on the part of the Fourier-Motzkin elimination that deals with them, and invite the reader to refer to Anderson et al. (2019) for the others. The combination of these new inequalities yields trivial constraints. For instance:

$$(34\text{i}) + (34\text{g}) \implies \hat{\mathbf{l}}_k[i] \leq \boldsymbol{w}^T_{i,k}\mathbf{x}_{k-1} + \boldsymbol{b}_k[i] = \hat{\mathbf{x}}_k[i] \tag{35}$$

which holds by the definition of pre-activation bounds.

Let us recall that $\mathbf{x}_k[i] \geq 0$ and $\mathbf{x}_k[i] \geq \hat{\mathbf{x}}_k[i]$, the latter constraint resulting from (34a) + (34b). Then, it can be easily verified that the only combinations of interest (i.e., those that do not result in constraints that are obvious by definition or are implied by other constraints) are those containing the equality (34a). In particular, combining inequalities (34g) to (34j) with inequalities (34d) to (34f) generates constraints that are (after algebraic manipulations) superfluous with respect to those in (36). We are now ready to show the system resulting from the elimination:

$$\mathbf{x}_k[i] \geq 0 \tag{36a}$$

$$\mathbf{x}_k[i] \geq \hat{\mathbf{x}}_k[i] \tag{36b}$$

$$\mathbf{x}_k[i] \leq \boldsymbol{w}_{i,k}[0]\mathbf{x}_{k-1}[0] - \boldsymbol{w}_{i,k}[0]\mathbf{l}_{k-1}[0](1-\mathbf{z}_k[i]) + \sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i] \tag{36c}$$

$$\mathbf{x}_k[i] \leq \boldsymbol{w}_{i,k}[0]\mathbf{u}_{k-1}[0]\mathbf{z}_k[i] + \sum_{j>1}\boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i] \tag{36d}$$

$$\mathbf{x}_k[i] \geq \boldsymbol{w}_{i,k}[0]\mathbf{x}_{k-1}[0] - \boldsymbol{w}_{i,k}[0]\mathbf{u}_{k-1}[0](1 - \mathbf{z}_k[i]) + \sum_{j>1} \boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i] \tag{36e}$$

$$\mathbf{x}_k[i] \geq \boldsymbol{w}_{i,k}[0]\mathbf{l}_{k-1}[0]\mathbf{z}_k[i] + \sum_{j>1} \boldsymbol{w}_{i,k}[j]\mathbf{x}^0_{k-1}[j] + \boldsymbol{b}_k[i]\mathbf{z}_k[i] \tag{36f}$$

$$\mathbf{l}_{k-1}[0] \leq \mathbf{x}_k[i] \leq \mathbf{u}_{k-1}[0] \tag{36g}$$

$$\mathbf{x}_k[i] \geq \hat{\mathbf{l}}_k[i]\mathbf{z}_k[i] \tag{36h}$$

$$\mathbf{x}_k[i] \leq \hat{\mathbf{u}}_k[i]\mathbf{z}_k[i] \tag{36i}$$

$$\mathbf{x}_k[i] \leq \hat{\mathbf{x}}_k[i] - \hat{\mathbf{l}}_k[i](1 - \mathbf{z}_k[i]) \tag{36j}$$

$$\mathbf{x}_k[i] \geq \hat{\mathbf{x}}_k[i] - \hat{\mathbf{u}}_k[i](1 - \mathbf{z}_k[i]) \tag{36k}$$

Constraints from (36a) to (36g) are those resulting from the original derivation of $\mathcal{A}'_k$ (see (Anderson et al., 2019)). The others result from the inclusion of pre-activation bounds in (33). Of these, (36h) is implied by (36a) if $\hat{\mathbf{l}}_k[i] \leq 0$ and by the definition of pre-activation bounds (together with (36b)) if $\hat{\mathbf{l}}_k[i] > 0$. Analogously, (36k) is implied by (36b) if $\hat{\mathbf{u}}_k[i] \geq 0$ and by (36a) otherwise.

By noting that no auxiliary variable is left in (36i) and in (36j), we can conclude that these will not be affected by the remaining part of the elimination procedure. Therefore, the rest of the proof (the elimination of $\mathbf{x}^0_{k-1}[1]$, $\mathbf{x}^0_{k-1}[2]$, $\dots$) proceeds as in (Anderson et al., 2019), leading to $\mathcal{A}_{k,i}$. Repeating the proof for each neuron i at layer k, we get $\mathcal{A}_k = \text{Proj}_{\mathbf{x}_{k-1},\mathbf{x}_k,\mathbf{z}_k}(\mathcal{S}_k)$.

■

Appendix G. Masked Forward and Backward Passes

Crucial to the practical efficiency of our solvers is to represent the various operations as standard forward/backward passes over a neural network. This way, we can leverage the engineering efforts behind popular deep learning frameworks such as PyTorch (Paszke et al., 2017). While this can be trivially done for the Big-M solver (appendix B), both the Active Set (§4) and Saddle Point (§5) solvers require a specialized operator that we call "masked" forward/backward pass. We now provide the details to our implementation.

As a reminder from §6, masked forward and backward passes respectively take the following forms (writing convolutional operators via their fully connected equivalents):

$$\left(W_k \odot I_k\right) \mathbf{a}_k, \quad \left(W_k \odot I_k\right)^T \mathbf{a}_{k+1},$$

where $\mathbf{a}_k \in \mathbb{R}^{n_k}, \mathbf{a}_{k+1} \in \mathbb{R}^{n_{k+1}}$. They are needed when dealing with the exponential family of constraints from the relaxation by Anderson et al. (2020).

Masked operators are straightforward to implement for fully connected layers (via element-wise products). We instead need to be more careful when handling convolutional layers. Standard convolution relies on re-applying the same weights (kernel) to many different parts of the image. A masked pass, instead, implies that the convolutional kernel is dynamically changing while it is being slided through the image. A naive solution is to convert convolutions into equivalent linear operators, but this has a high cost in terms of performance, as it involves much redundancy. Our implementation relies on an alternative view of convolutional layers, which we outline next.

G.1 Convolution as Matrix-matrix Multiplication

A convolutional operator can be represented via a matrix-matrix multiplication if the input is *unfolded* and the filter is appropriately reshaped. The multiplication output can then be reshaped to the correct convolutional output shape. Given a filter $w \in \mathbb{R}^{c \times k_1 \times k_2}$, an input $\mathbf{x} \in \mathbb{R}^{i_1 \times i_2 \times i_3}$ and the convolutional output $\texttt{conv}_w(\mathbf{x}) = \mathbf{y} \in \mathbb{R}^{c \times o_2 \times o_3}$, we need the following definitions:

$$\begin{aligned} [\cdot]_{\mathcal{I},\mathcal{O}} &: \mathcal{I} \rightarrow \mathcal{O} \\ \{\cdot\}_j &: \mathbb{R}^{d_1 \times \cdots \times d_n} \rightarrow \mathbb{R}^{d_1 \times \cdots \times d_{j-1} \times d_{j+1} \times \cdots \times d_n} \\ \texttt{unfold}_w(\cdot) &: \mathbb{R}^{i_1 \times i_2 \times i_3} \rightarrow \mathbb{R}^{k_1 k_2 \times o_2 o_3} \\ \texttt{fold}_w(\cdot) &: \mathbb{R}^{k_1 k_2 \times o_2 o_3} \rightarrow \mathbb{R}^{i_1 \times i_2 \times i_3} \end{aligned} \tag{37}$$

where the brackets simply reshape the vector from shape $\mathcal{I}$ to $\mathcal{O}$, while the braces sum over the j-th dimension. `unfold` decomposes the input image into the (possibly overlapping) $o_2 o_3$ blocks the sliding kernel operates on, taking padding and striding into account. `fold` brings the output of `unfold` to the original input space.

Let us define the following reshaped versions of the filter and the convolutional output:

$$\begin{aligned} W_R &= [w]_{\mathbb{R}^{c \times k_1 \times k_2}, \mathbb{R}^{c \times k_1 k_2}} \\ \mathbf{y}_R &= [\mathbf{y}]_{\mathbb{R}^{c \times o_2 \times o_3}, \mathbb{R}^{c \times o_2 o_3}} \end{aligned}$$

The standard forward/backward convolution (neglecting the convolutional bias, which can be added at the end of the forward pass) can then be written as:

$$\begin{aligned} \texttt{conv}_w(\mathbf{x}) &= [W_R \ \texttt{unfold}_w(\mathbf{x})]_{\mathbb{R}^{c \times o_2 o_3}, \mathbb{R}^{c \times o_2 \times o_3}} \\ \texttt{back\_conv}_w(\mathbf{y}) &= \texttt{fold}_w(W_R^T \ \mathbf{y}_R). \end{aligned} \tag{38}$$

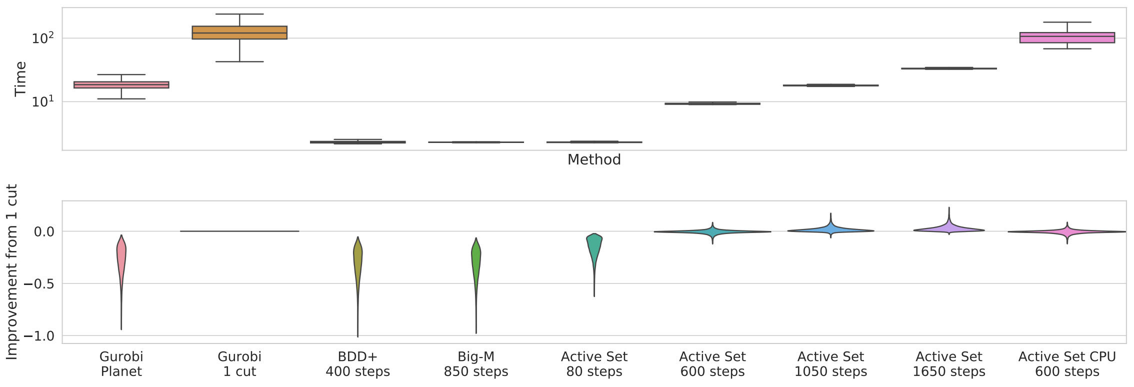
G.2 Masked Convolution as Matrix-matrix Multiplication

We need to mask the convolution with a different scalar for each input-output pair. Therefore, we employ a mask $I \in \mathbb{R}^{c \times k_1 k_2 \times o_2 o_3}$, whose additional dimension with respect to W_R is associated to the output space of the convolution. Assuming vectors are broadcast to the correct output shape[6] , we can write the masked forward and backward passes by adapting equation (38) as follows:

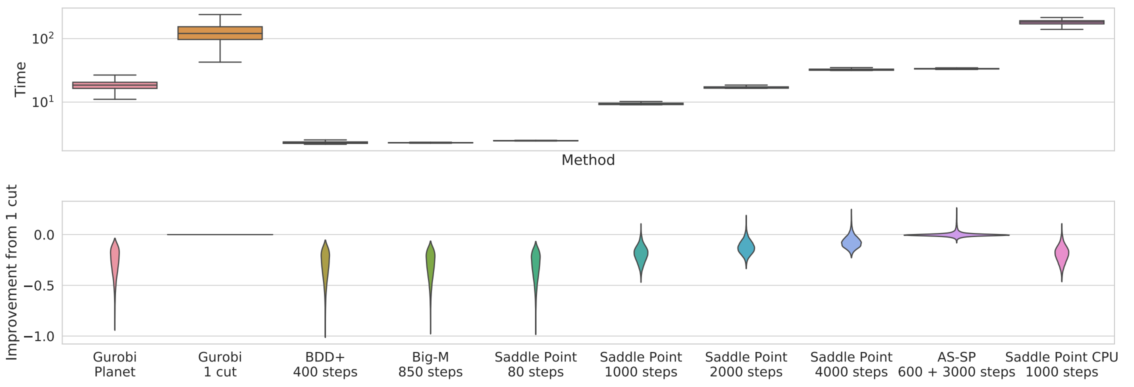
$$\begin{aligned} \texttt{conv}_{w,I}(\mathbf{x}) &= [\{W_R \ \odot I \odot \texttt{unfold}_w(\mathbf{x})\}_2]_{\mathbb{R}^{c \times o_2 o_3}, \mathbb{R}^{c \times o_2 \times o_3}} \\ \texttt{back\_conv}_{w,I}(\mathbf{y}) &= \texttt{fold}_w(\{W_R \ \odot I \odot \mathbf{y}_R\}_1). \end{aligned} \tag{39}$$

6. if we want to perform an element-wise product $\mathbf{a} \odot \mathbf{b}$ between $\mathbf{a} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ and $\mathbf{b} \in \mathbb{R}^{d_1 \times d_3}$, the operation is implicitly performed as $\mathbf{a} \odot \mathbf{b}'$, where $\mathbf{b}' \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ is an extended version of $\mathbf{b}$ obtained by copying along the missing dimension.

Owing to the avoided redundancy with respect to the equivalent linear operation (e.g., copying of the kernel matrix, zero-padding in the linear weight matrix), this implementation of the masked forward/backward pass reduces both the memory footprint and the number of floating point operations (FLOPs) associated to the passes computations by a factor $(i_1 i_2 i_3)/(k_1 k_2)$. In practice, this ratio might be significant: on the incomplete verification networks (§8.2) it ranges from 16 to 64 depending on the layer.

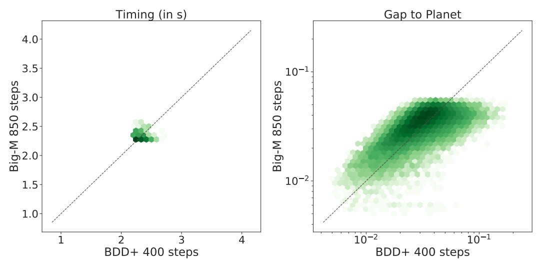


(a) Speed-accuracy trade-offs of Active Set for different iteration ranges and computing devices.

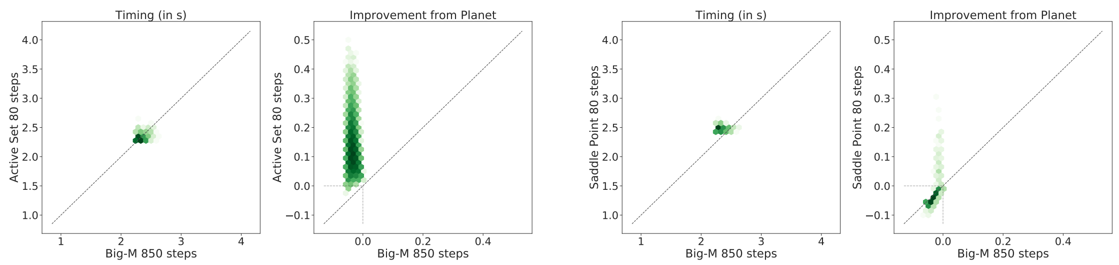


(b) Speed-accuracy trade-offs of Saddle Point for different iteration ranges and computing devices.

Figure 9: Upper bounds to the adversarial vulnerability for the network adversarially trained with the method by Madry et al. (2018), from Bunel et al. (2020a). Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better, the width at a given value represents the proportion of problems for which this is the result.



(a) Comparison of runtime (left) and gap to Gurobi Planet bounds (right). For the latter, lower is better.



(b) Comparison of runtime (Timing) and difference with the Gurobi Planet bounds (Improvement from Planet). For the latter, higher is better.

Figure 10: Pointwise comparison for a subset of the methods on the data presented in figure 1. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

Appendix H. Experimental Appendix

We conclude the appendix by presenting supplementary experiments with respect to the presentation in the main paper.

H.1 Adversarially-Trained CIFAR-10 Incomplete Verification

In addition to the SGD-trained network in §8.2, we now present results relative to the same architecture, trained with the adversarial training method by Madry et al. (2018) for robustness to perturbations of $\epsilon_{train} = 2/255$. Each adversarial sample for the training was obtained using 50 steps of projected gradient descent. For this network, we upper bound the vulnerability to perturbations with $\epsilon_{ver} = 2.7/255$. Hyper-parameters are kept to the values tuned on the SGD-trained network from Section 8.2.

Figures 9, 10, 11 confirm most of the observations carried out for the SGD-trained network in §8.2, with fewer variability around the bounds returned by Gurobi cut. Big-M is competitive with BDD+, and switching to Active Set after 500 iterations results in much better bounds in the same time. Increasing the computational budget for Active Set still results in better bounds than Gurobi cut in a fraction of its running time, even though the performance gap is on average smaller than on the SGD-trained network. As in Section 8.2 the gap between Saddle Point and Active Set, though larger here on average, decreases with the computational budget, and is further reduced when initializing with a few Active Set iterations.

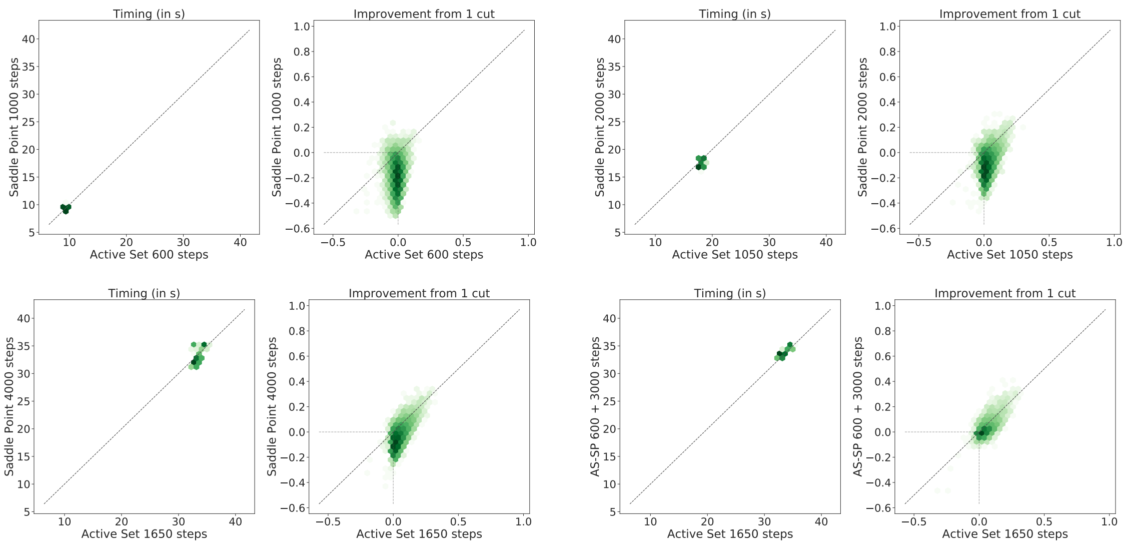


Figure 11: Pointwise comparison between our proposed solvers on the data presented in figure 9. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

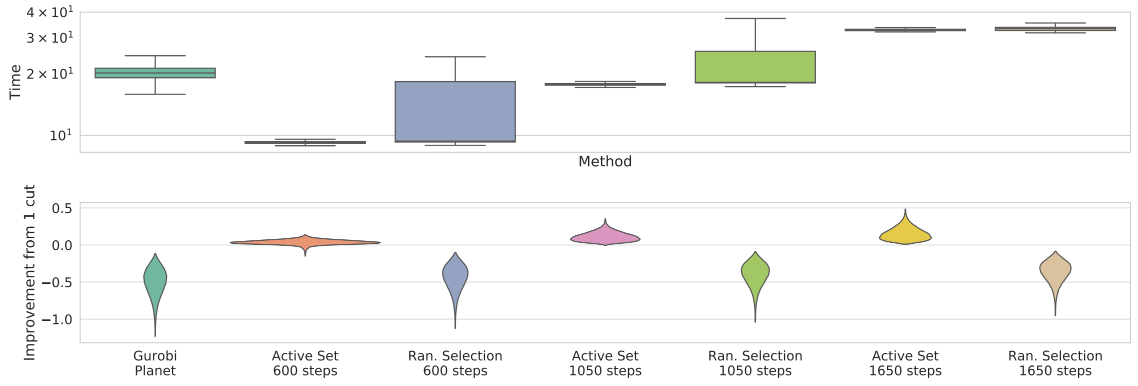


Figure 12: Upper plot: distribution of runtime in seconds. Lower plot: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. Results for the SGD-trained network from Bunel et al. (2020a). Sensitivity of Active Set to selection criterion (see §4.2).

H.2 Sensitivity of Active Set to Selection Criterion and Frequency

In Section 4.2, we describe how to iteratively modify $\mathcal{B}$, the active set of dual variables on which our Active Set solver operates. In short, Active Set adds the variables corresponding to the output of oracle (4) invoked at the primal minimiser of $\mathcal{L}_{\mathcal{B}}(\mathbf{x}, \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}_{\mathcal{B}})$, at a fixed frequency ω. We now investigate the empirical sensitivity of Active Set to both the selection criterion and the frequency of addition.

We test against **Ran. Selection**, a version of Active Set for which the variables to add are selected at random by uniformly sampling from the binary I_k masks. As expected, Figure 12 shows that a good selection criterion is key to the efficiency of Active Set. In fact, random variable selection only marginally improves upon the Planet relaxation bounds, whereas the improvement becomes significant with our selection criterion from §4.2.

In addition, we investigate the sensitivity of Active Set (AS) to variable addition frequency ω. In order to do so, we cap the maximum number of cuts at 7 for all runs, and vary ω while keeping the time budget fixed (we test on three different time budgets). Figure 13 compares the results for $\omega = 450$ (Active Set), which were presented in §8.2, with the bounds obtained by setting $\omega = 300$ and $\omega = 600$ (respectively **AS $\boldsymbol{\omega} = $ 300** and **AS $\boldsymbol{\omega} = $ 600**). Our solver proves to be relatively robust to ω across all the three budgets, with the difference in obtained bounds decreasing with the number of iterations. Moreover, early cut addition tends to yield better bounds in the same time, suggesting that our selection criterion is effective before subproblem convergence.

H.3 MNIST Incomplete Verification

We conclude the experimental appendix by presenting incomplete verification results (the experimental setup mirrors the one employed in Section 8.2 and appendix H.1) on the MNIST dataset (LeCun et al., 1998).

We report results on the "wide" MNIST network from Lu and Kumar (2020), whose architecture is identical to the "wide" network in Table 1 except for the first layer, which has only one input channel to reflect the MNIST specification (the total number of ReLU activation units is 4804). As those employed for the complete verification experiments (§8.3), and differently from the incomplete verification experiments in Section 8.2 and appendix H.1, the network was trained with the verified training method by Wong and Kolter (2018).

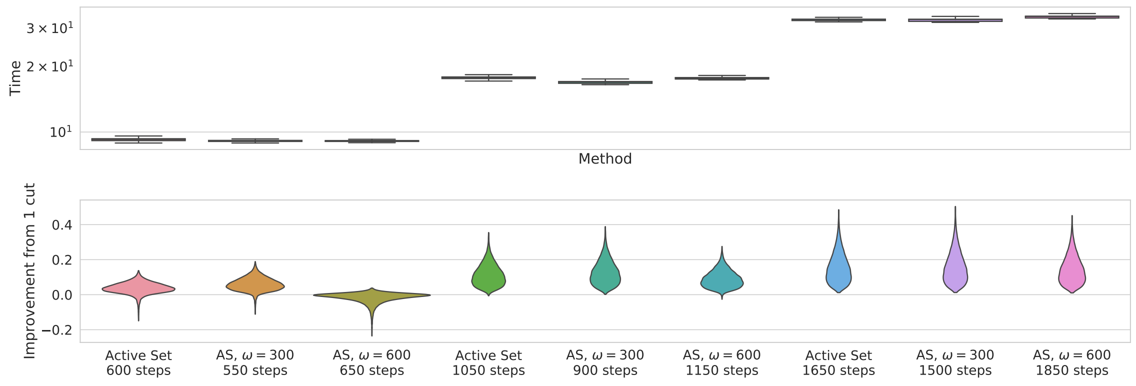
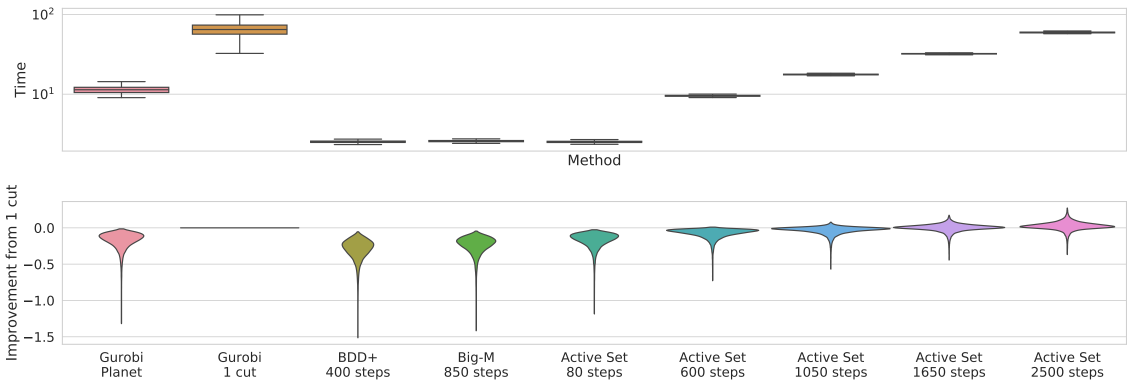
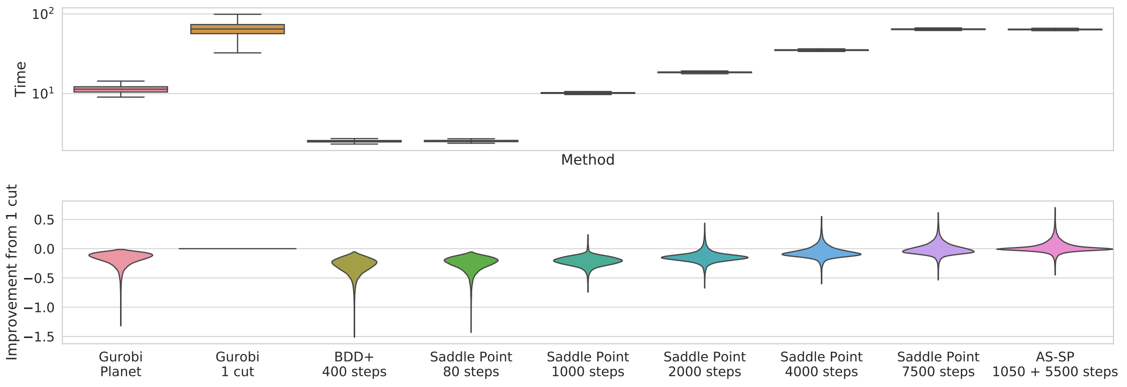


Figure 13: Upper plot: distribution of runtime in seconds. Lower plot: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better. Results for the SGD-trained network from Bunel et al. (2020a). Sensitivity of Active Set to variable addition frequency ω, with the selection criterion presented in §4.2.



(a) Speed-accuracy trade-offs of Active Set for different iteration ranges and computing devices.



(b) Speed-accuracy trade-offs of Saddle Point for different iteration ranges and computing devices.

Figure 14: Upper bounds to the adversarial vulnerability for the MNIST network trained with the verified training algorithm by Wong and Kolter (2018), from Lu and Kumar (2020). Box plots: distribution of runtime in seconds. Violin plots: difference with the bounds obtained by Gurobi with a cut from $\mathcal{A}_k$ per neuron; higher is better.

We compute the vulnerability to $\epsilon_{ver} = 0.15$ on the first 820 images of the MNIST test set. All hyper-parameters are kept to the values employed for the CIFAR-10 networks, except the Big-M step size, which was linearly decreased from 10^{-1} to 10^{-3}, and the weight of the proximal terms for BDD+, which was linearly increased from 1 to 50.

As seen on the CIFAR-10 networks, Figures 14, 15 show that our solvers for problem (3) (Active Set and Saddle Point) yield comparable or better bounds than Gurobi 1 cut in less average runtime. However, more iterations are required to reach the same relative bound improvement over Gurobi 1 cut (for Active Set, 2500 as opposed to 600 in Figures 1, 9). Finally, the smaller average gap between the bounds of Gurobi Planet and Gurobi 1 cut (especially with respect to Figure 1) suggests that the relaxation by Anderson et al. (2020) is less effective on this MNIST benchmark.

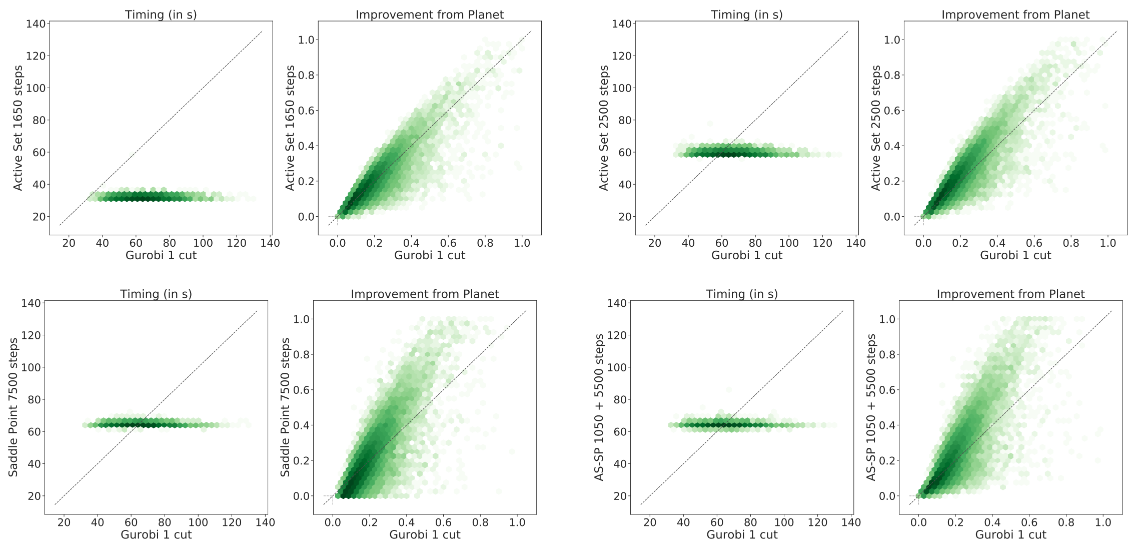


Figure 15: Pointwise comparison for a subset of the methods on the data presented in Figure 14. Comparison of runtime (left) and improvement from the Gurobi Planet bounds. For the latter, higher is better. Darker colour shades mean higher point density (on a logarithmic scale). The oblique dotted line corresponds to the equality.

References

Ross Anderson, Joey Huchette, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. In *Conference on Integer Programming and Combinatorial Optimization*, 2019.

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 2020.

Mislav Balunovic and Martin Vechev. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations*, 2020.

Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of ReLU-based neural networks via dependency analysis. In *AAAI Conference on Artificial Intelligence*, 2020.

Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. A unified view of piecewise linear neural network verification. In *Neural Information Processing Systems*, 2018.

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

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

Emmanuel J. Candès, Michael B. Wakin, and Stephen P. Boyd. Enhancing sparsity by reweighted ℓ 1 minimization. *Journal of Fourier Analysis and Applications*, 2008.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy*, 2017.

Alessandro De Palma, Harkirat Singh Behl, Rudy Bunel, Philip H. S. Torr, and M. Pawan Kumar. Scaling the convex barrier with active sets. In *International Conference on Learning Representations*, 2021.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Uncertainty in Artificial Intelligence*, 2018.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Chongli Qin, Soham De, and Pushmeet Kohli. Efficient neural network verification with exactness characterization. In *Uncertainty in Artificial Intelligence*, 2020.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. *Automated Technology for Verification and Analysis*, 2017.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 1956.

Gauthier Gidel, Tony Jebara, and Simon Lacoste-Julien. Frank-Wolfe algorithms for saddle point problems. In *International Conference on Artificial Intelligence and Statistics*, 2017.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. In *SECML NeurIPS*, 2018.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. URL `http://www.gurobi.com`.

Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *International Conference on Machine Learning*, 2013.

R. G. Jeroslow. Representability in mixed integer programming, i: Characterization results. *Discrete Applied Mathematics*, 1987.

Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer-Aided Verification*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations*, 2017.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *IEEE*, 1998.

Claude Lemaréchal. Lagrangian relaxation. In *Computational combinatorial optimization*, pages 112–156. Springer, 2001.

Jingyue Lu and M Pawan Kumar. Neural network branching for neural network verification. In *International Conference on Learning Representations*, 2020.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS Autodiff Workshop*, 2017.

Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Neural Information Processing Systems*, 2018.

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

Hanif D. Sherali and Gyunghyun Choi. Recovery of primal solutions when using subgradient optimization methods to solve lagrangian duals of linear programs. *Operations Research Letters*, 1996.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Neural Information Processing Systems*, 2018.

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

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*, 2019b.

Gagandeep Singh, Jonathan Maurer, Christoph Müller, Matthew Mirman, Timon Gehr, Adrian Hoffmann, Petar Tsankov, Dana Drachsler Cohen, Markus Püschel, and Martin Vechev. ETH robustness analyzer for neural networks (ERAN). 2020. URL `https://github.com/eth-sri/eran`.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Patel, and Juan Pablo Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *arXiv preprint arXiv:2006.14076*, 2020.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019.

VNN-COMP. International verification of neural networks competition (VNN-COMP). *Verification of Neural Networks workshop at the International Conference on Computer-Aided Verification*, 2020. URL `https://sites.google.com/view/vnn20/vnncomp`.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Neural Information Processing Systems*, 2018.

Stefan Webb, Tom Rainforth, Yee Whye Teh, and M Pawan Kumar. A statistical approach to assessing neural network robustness. In *International Conference on Learning Representations*, 2019.

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

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

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021.