Lagrangian Decomposition for Neural Network Verification

Rudy Bunel*, **Alessandro De Palma***, **Alban Desmaison**
University of Oxford
{rudy, adepalma}@robots.ox.ac.uk

Krishnamurthy (Dj) Dvijotham, Pushmeet Kohli
DeepMind

Philip H.S. Torr, M. Pawan Kumar
University of Oxford

Abstract

A fundamental component of neural network verification is the computation of bounds on the values their outputs can take. Previous methods have either used off-the-shelf solvers, discarding the problem structure, or relaxed the problem even further, making the bounds unnecessarily loose. We propose a novel approach based on Lagrangian Decomposition. Our formulation admits an efficient supergradient ascent algorithm, as well as an improved proximal algorithm. Both the algorithms offer three advantages: (i) they yield bounds that are provably at least as tight as previous dual algorithms relying on Lagrangian relaxations; (ii) they are based on operations analogous to forward/backward pass of neural networks layers and are therefore easily parallelizable, amenable to GPU implementation and able to take advantage of the convolutional structure of problems; and (iii) they allow for anytime stopping while still providing valid bounds. Empirically, we show that we obtain bounds comparable with off-the-shelf solvers in a fraction of their running time, and obtain tighter bounds in the same time as previous dual algorithms. This results in an overall speed-up when employing the bounds for formal verification. Code for our algorithms is available at https://github.com/oval-group/decomposition-plnn-bounds.

1 INTRODUCTION

As deep learning powered systems become more and more common, the lack of robustness of neural networks and their reputation for being "Black Boxes" is increasingly worrisome. In order to deploy them in critical scenarios where safety and robustness would be a prerequisite, we need to invent techniques that can prove formal guarantees for neural network behaviour. A particularly desirable property is resistance to adversarial examples (Goodfellow et al., 2015, Szegedy et al., 2014): perturbations maliciously crafted with the intent of fooling even extremely well performing models. After several defenses were proposed and subsequently broken (Athalye et al., 2018, Uesato et al., 2018), some progress has been made in being able to formally verify whether there exist any adversarial examples in the neighbourhood of a data point (Tjeng et al., 2019, Wong and Kolter, 2018).

Verification algorithms fall into three categories: unsound (some false properties are proven false), incomplete (some true properties are proven true), and complete (all properties are correctly verified as either true or false). A critical component of the verification systems developed so far is the computation of lower and upper bounds on the output of neural networks when their inputs are constrained to lie in a bounded set. In incomplete verification, by deriving bounds on the changes of the prediction vector under restricted perturbations, it is possible to identify safe regions of the input space. These results allow the rigorous comparison of adversarial defenses and prevent making overconfident statements about their efficacy (Wong and Kolter, 2018). In complete verification, bounds can also be used as essential subroutines of Branch and Bound complete verifiers (Bunel et al., 2018). Finally, bounds might also be used as a training signal to guide the network towards greater robustness and more verifiability (Gowal et al., 2018, Mirman et al., 2018, Wong and Kolter, 2018).

Most previous algorithms for computing bounds are either computationally expensive (Ehlers, 2017) or sacrifice a lot of tightness in order to scale (Gowal et al., 2018, Mirman et al., 2018, Wong and Kolter, 2018). In this work, we design novel customised relaxations and their corresponding solvers for obtaining bounds on neural networks. Our approach offers the following advantages:

* These authors contributed equally to this work.

Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), PMLR volume 124, 2020.

- While previous approaches to neural network bounds (Dvijotham et al., 2018) are based on Lagrangian relaxations, we derive a new family of optimization problems for neural network bounds through **Lagrangian Decomposition**, which in general yields duals at least as strong as those obtained through Lagrangian relaxation (Guignard and Kim, 1987). We in fact prove that, in the context of ReLU networks, for any dual solution from the approach by Dvijotham et al. (2018), the bounds output by our dual are as least as tight. We demonstrate empirically that this derivation computes tighter bounds in the same time when using supergradient methods. We further improve on the performance by devising a proximal solver for the problem, which decomposes the task into a series of strongly convex subproblems. For each, we use an iterative method for which we derive optimal step sizes.
- Both the supergradient and the proximal method operate through linear operations similar to those used during network forward/backward passes. As a consequence, we can **leverage the convolutional structure** when necessary, while standard solvers are often restricted to treating it as a general linear operation. Moreover, both methods are **easily parallelizable**: when computing bounds on the activations at layer k, we need two solve two problems for each hidden unit of the network (for the upper and lower bounds). These can all be solved in parallel. In complete verification, we need to compute bounds for several different problem domains at once: we solve these problems in parallel as well. Our GPU implementation thus allows us to solve several hundreds of linear programs at once on a single GPU, a level of parallelism that would be hard to match on CPU-based systems.
- Most standard linear programming based relaxations (Ehlers, 2017) will only return a valid bound if the problem is solved to optimality. Others, like the dual simplex method employed by off-the-shelf solvers (Gurobi Optimization, 2020) have a very high cost per iteration and will not yield tight bounds without incurring significant computational costs. Both methods described in this paper are **anytime** (terminating it before convergence still provides a valid bound), and can be interrupted at very small granularity. This is useful in the context of a subroutine for complete verifiers, as this enables the user to choose an appropriate speed versus accuracy trade-off. It also offers great versatility as an incomplete verification method.

2 RELATED WORKS

Bound computations are mainly used for formal verification methods. Some methods are complete (Cheng et al., 2017, Ehlers, 2017, Katz et al., 2017, Tjeng et al., 2019, Xiang et al., 2017), always returning a verdict for each problem instances. Others are incomplete, based on relaxations of the verification problem. They trade speed for completeness: while they cannot verify properties for all problem instances, they scale significantly better. Two main types of bounds have been proposed: on the one hand, some approaches (Ehlers, 2017, Salman et al., 2019) rely on off-the-shelf solvers to solve accurate relaxations such as PLANET (Ehlers, 2017), which is the best known linear-sized approximation of the problem. On the other hand, as PLANET and other more complex relaxations do not have closed form solutions, some researchers have also proposed easier to solve, looser formulations (Gowal et al., 2018, Mirman et al., 2018, Weng et al., 2018, Wong and Kolter, 2018). Explicitly or implicitly, these are all equivalent to propagating a convex domain through the network to overapproximate the set of reachable values. Our approach consists in tackling a relaxation equivalent to the PLANET one (although generalised beyond ReLU), by designing a custom solver that achieves faster performance without sacrificing tightness. Some potentially tighter convex relaxations exist but involve a quadratic number of variables, such as the semi-definite programming method of Raghunathan et al. (2018) . Better relaxations obtained from relaxing strong Mixed Integer Programming formulations (Anderson et al., 2019) have a quadratic number of variables or a potentially exponential number of constraints. We do not address them here.

A closely related approach to ours is the work of Dvijotham et al. (2018). Both their method and ours are anytime and operate on similar duals. While their dual is based on the Lagrangian relaxation of the non-convex problem, ours is based on the Lagrangian Decomposition of the nonlinear activation's convex relaxation. Thanks to the properties of Lagrangian Decomposition (Guignard and Kim, 1987), we can show that our dual problem provides better bounds when evaluated on the same dual variables. The relationship between the two duals is studied in detail in section 4.2. Moreover, in terms of the followed optimization strategy, in addition to using a supergradient method like Dvijotham et al. (2018), we present a proximal method, for which we can derive optimal step sizes. We show that these modifications enable us to compute tighter bounds using the same amount of compute time.

3 PRELIMINARIES

Throughout this paper, we will use bold lower case letters (like $\mathbf{z}$) to represent vectors and upper case letters (like W) to represent matrices. Brackets are used to indicate the i-th coordinate of a vector ($\mathbf{z}[i]$), and integer ranges (e.g., $[1, n-1]$). We will study the computation of the lower bound problem based on a feedforward neural network, with element-wise activation function $\sigma(\cdot)$. The network inputs are restricted to a convex domain $\mathcal{C}$, over which we assume that we can easily optimise linear functions. This

is the same assumption that was made by Dvijotham et al. (2018). The computation for an upper bound is analogous. Formally, we wish to solve the following problem:

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

The output of the k-th layer of the network before and after the application of the activation function are denoted by $\hat{\mathbf{z}}_k$ and $\mathbf{z}_k$ respectively. Constraints (1c) implements the linear layers (fully connected or convolutional) while constraints (1d) implement the non-linear activation function. Constraints (1b) define the region over which the bounds are being computed. While our method can be extended to more complex networks (such as ResNets), we focus on problem (1) for the sake of clarity.

The difficulty of problem (1) comes from the non-linearity (1d). Dvijotham et al. (2018) tackle it by relaxing (1c) and (1b) via Lagrangian multipliers, yielding the following dual:

$$\begin{aligned}
&\max_{\boldsymbol{\mu},\boldsymbol{\lambda}} \; d(\boldsymbol{\mu},\boldsymbol{\lambda}), \quad \text{where:}\\
d(\boldsymbol{\mu},\boldsymbol{\lambda}) = &\min_{\mathbf{z},\hat{\mathbf{z}}} \quad W_n\mathbf{z}_{n-1} + \mathbf{b}_n\\
&+ \sum_{k=1}^{n-1} \boldsymbol{\mu}_k^T \left(\hat{\mathbf{z}}_k - W_k\mathbf{z}_{k-1} - \mathbf{b}_k\right)\\
&+ \sum_{k=1}^{n-1} \boldsymbol{\lambda}_k^T \left(\mathbf{z}_k - \sigma(\hat{\mathbf{z}}_k)\right)\\
&\text{s.t.} \quad \mathbf{l}_k \le \hat{\mathbf{z}}_k \le \mathbf{u}_k \qquad k \in [1, n-1]\,,\\
&\qquad \sigma(\mathbf{l}_k) \le \mathbf{z}_k \le \sigma(\mathbf{u}_k) \qquad k \in [1, n-1]\,,\\
&\qquad \mathbf{z}_0 \in \mathcal{C}.
\end{aligned} \tag{2}$$

If σ is a ReLU, this relaxation is equivalent (Dvijotham et al., 2018) to the PLANET relaxation (Ehlers, 2017). The dual requires upper ($\mathbf{u}_k$) and lower bounds ($\mathbf{l}_k$) on the value that $\hat{\mathbf{z}}_k$ can take, for $k \in [0..n-1]$. We call these *intermediate bounds*: we detail how to compute them in appendix B. The dual algorithm by Dvijotham et al. (2018) solves (2) via supergradient ascent.

4 LAGRANGIAN DECOMPOSITION

We will now describe a novel approach to solve problem (1), and relate it to the dual algorithm by Dvijotham et al. (2018). We will focus on computing bounds for an output of the last layer of the neural network.

4.1 PROBLEM DERIVATION

Our approach is based on Lagrangian decomposition, also known as variable splitting (Guignard and Kim, 1987). Due to the compositional structure of neural networks, most constraints involve only a limited number of variables. As a result, we can split the problem into meaningful, easy to solve subproblems. We then impose constraints that the solutions of the subproblems should agree.

We start from the original non-convex primal problem (1), and substitute the nonlinear activation equality (1d) with a constraint corresponding to its convex hull. This leads to the following convex program:

$$\begin{aligned}
\min_{\mathbf{z},\hat{\mathbf{z}}} \quad & \hat{\mathbf{z}}_n && \text{(3a)}\\
\text{s.t.} \quad & \mathbf{z}_0 \in \mathcal{C}, && \text{(3b)}\\
& \hat{\mathbf{z}}_{k+1} = W_{k+1}\mathbf{z}_k + \mathbf{b}_{k+1} \quad k \in [1, n-1]\,, && \text{(3c)}\\
& \texttt{cvx_hull}_\sigma\left(\hat{\mathbf{z}}_k, \mathbf{z}_k, \mathbf{l}_k, \mathbf{u}_k\right) \quad k \in [1, n-1]\,. && \text{(3d)}
\end{aligned}$$

In the following, we will use ReLU activation functions as an example, and employ the PLANET relaxation as its convex hull, which is the tightest linearly sized relaxation published thus far. By linearly sized, we mean that the numbers of variables and constraints to describe the relaxation only grow linearly with the number of units in the network. We stress that the derivation can be extended to other non-linearities. For example, appendix A describes the case of sigmoid activation function. For ReLUs, this convex hull takes the following form (Ehlers, 2017):

$$\texttt{cvx_hull}_\sigma \equiv \begin{cases}
\text{if } \mathbf{l}_k[i] \le 0; \mathbf{u}_k[i] \ge 0: & \\
\quad \mathbf{z}_k[i] \ge 0, \quad \mathbf{z}_k[i] \ge \hat{\mathbf{z}}_k[i], & \text{(4a)}\\
\quad \mathbf{z}_k[i] \le \dfrac{\mathbf{u_k}[i](\hat{\mathbf{z}}_k[i] - \mathbf{l}_k[i])}{\mathbf{u_k}[i] - \mathbf{l_k}[i]}. & \\
\text{if } \mathbf{u}_k[i] \le 0: & \\
\quad \mathbf{z}_k[i] = 0. & \text{(4b)}\\
\text{if } \mathbf{l}_k[i] \ge 0: & \\
\quad \mathbf{z}_k[i] = \hat{\mathbf{z}}_k[i]. & \text{(4c)}
\end{cases}$$

Constraints (4a), (4b), and (4c) corresponds to the relaxation of the ReLU (1d) in different cases (respectively: ambiguous state, always blocking or always passing).

To obtain a decomposition, we divide the constraints into subsets. Each subset will correspond to a pair of an activation layer, and the linear layer coming after it. The only exception is the first linear layer which is combined with the restriction of the input domain to $\mathcal{C}$. This choice is motivated by the fact that for piecewise-linear activation functions, we will be able to easily perform linear optimisation over the resulting subdomains. For a different activation function, it might be required to have a different decomposition. Formally, we introduce the following notation for subsets of constraints:

$$\mathcal{P}_0(\mathbf{z}_0, \hat{\mathbf{z}}_1) \equiv \begin{cases} \mathbf{z}_0 \in \mathcal{C} \\ \hat{\mathbf{z}}_1 = W_1\mathbf{z}_0 + \mathbf{b}_1, \end{cases} \tag{5}$$

$$\mathcal{P}_k(\hat{\mathbf{z}}_k, \hat{\mathbf{z}}_{k+1}) \equiv \begin{cases} \mathbf{l}_k \le \hat{\mathbf{z}}_k \le \mathbf{u}_k, \\ \texttt{cvx\_hull}_\sigma\left(\hat{\mathbf{z}}_k, \mathbf{z}_k, \mathbf{l}_k, \mathbf{u}_k\right), \\ \hat{\mathbf{z}}_{k+1} = W_{k+1}\mathbf{z}_k + \mathbf{b}_k. \end{cases} \tag{6}$$

The set $\mathcal{P}_k$ is defined without an explicit dependency on $\mathbf{z}_k$ because $\mathbf{z}_k$ only appears internally to the subset of

constraints and is not involved in the objective function. Using this grouping of the constraints, we can concisely write problem (3) as:

$$\min_{\mathbf{z},\hat{\mathbf{z}}} \quad \hat{\mathbf{z}}_n \qquad \text{s.t.} \quad \mathcal{P}_0(\mathbf{z}_0, \hat{\mathbf{z}}_1), \quad \mathcal{P}_k(\hat{\mathbf{z}}_k, \hat{\mathbf{z}}_{k+1}) \qquad k \in [1, n-1]\,. \tag{7}$$

To obtain a Lagrangian decomposition, we duplicate the variables so that each subset of constraints has its own copy of the variables it is involved in. Formally, we rewrite problem (7) as follows:

$$\min_{\mathbf{z},\hat{\mathbf{z}}} \quad \hat{\mathbf{z}}_{A,n} \tag{8a}$$

$$\text{s.t.} \quad \mathcal{P}_0(\mathbf{z}_0, \hat{\mathbf{z}}_{A,1}), \tag{8b}$$

$$\mathcal{P}_k(\hat{\mathbf{z}}_{B,k}, \hat{\mathbf{z}}_{A,k+1}) \qquad k \in [1, n-1]\,, \tag{8c}$$

$$\hat{\mathbf{z}}_{A,k} = \hat{\mathbf{z}}_{B,k} \qquad k \in [1, n-1]\,. \tag{8d}$$

The additional equality constraints (8d) impose agreements between the various copies of variables. We introduce the dual variables $\boldsymbol{\rho}$ and derive the Lagrangian dual:

$$\begin{aligned} &\max_{\boldsymbol{\rho}} \; q(\boldsymbol{\rho}), \quad \text{where:} \\ &q(\boldsymbol{\rho}) = \min_{\mathbf{z},\hat{\mathbf{z}}} \; \hat{\mathbf{z}}_{A,n} + \sum_{k=1}^{n-1} \boldsymbol{\rho}_k^T \left(\hat{\mathbf{z}}_{B,k} - \hat{\mathbf{z}}_{A,k}\right) \\ &\text{s.t.} \quad \mathcal{P}_0(\mathbf{z}_0, \hat{\mathbf{z}}_{A,1}), \\ &\qquad \mathcal{P}_k(\hat{\mathbf{z}}_{B,k}, \hat{\mathbf{z}}_{A,k+1}) \qquad k \in [1, n-1]\,. \end{aligned} \tag{9}$$

Any value of $\boldsymbol{\rho}$ provides a valid lower bound by virtue of weak duality. While we will maximise over the choice of dual variables in order to obtain as tight a bound as possible, we will be able to interrupt the optimisation process at any point and obtain a valid bound by evaluating q.

We stress that, at convergence, problems (9) and (2) will yield the same bounds, as they are both reformulations of the same convex problem. This does not imply that the two derivations yield solvers with the same efficiency. In fact, we will next prove that, for ReLU-based networks, problem (9) yields bounds at least as tight as a corresponding dual solution from problem (2).

4.2 DUALS COMPARISON

We now compare our dual problem (9) with problem (2) by Dvijotham et al. (2018). From the high level perspective, our decomposition considers larger subsets of constraints and hence results in a smaller number of variables to optimize. We formally prove that, for ReLU-based neural networks, our formulation dominates theirs, producing tighter bounds based on the same dual variables.

Theorem 1. *Let us assume $\sigma(\hat{\mathbf{z}}_k) = \max(\mathbf{0}, \hat{\mathbf{z}}_k)$. For any solution $\boldsymbol{\mu}, \boldsymbol{\lambda}$ of dual* (2) *by Dvijotham et al. (2018) yielding bound $d(\boldsymbol{\mu}, \boldsymbol{\lambda})$, it holds that $q(\boldsymbol{\mu}) \geq d(\boldsymbol{\mu}, \boldsymbol{\lambda})$.*

Proof. See appendix E. □

Our dual is also related to the one that Wong and Kolter (2018) operates in. We show in appendix F how our problem can be initialised to a set of dual variables so as to generate a bound matching the one provided by their algorithm. We use this approach as our initialisation for incomplete verification (§ 6.3).

5 SOLVERS FOR LAGRANGIAN DECOMPOSITION

Now that we motivated the use of dual (9) over problem (2), it remains to show how to solve it efficiently in practice. We present two methods: one based on supergradient ascent, the other on proximal maximisation. A summary, including pseudo-code, can be found in appendix D.

5.1 SUPERGRADIENT METHOD

As Dvijotham et al. (2018) use supergradient methods on their dual, we start by applying it on problem (9) as well.

At a given point $\boldsymbol{\rho}$, obtaining the supergradient requires us to know the values of $\hat{\mathbf{z}}_A$ and $\hat{\mathbf{z}}_B$ for which the inner minimisation is achieved. Based on the identified values of $\hat{\mathbf{z}}_A^*$ and $\hat{\mathbf{z}}_B^*$, we can then compute the supergradient $\nabla_{\boldsymbol{\rho}} q = \hat{\mathbf{z}}_B^* - \hat{\mathbf{z}}_A^*$. The updates to $\boldsymbol{\rho}$ are then the usual supergradient ascent updates:

$$\boldsymbol{\rho}^{t+1} = \boldsymbol{\rho}^t + \alpha^t \nabla_{\boldsymbol{\rho}} q(\boldsymbol{\rho}^t), \tag{10}$$

where α^t corresponds to a step size schedule that needs to be provided. It is also possible to use any variants of gradient descent, such as Adam (Kingma and Ba, 2015).

It remains to show how to perform the inner minimization over the primal variables. By design, each of the variables is only involved in one subset of constraints. As a result, the computation completely decomposes over the subproblems, each corresponding to one of the subset of constraints. We therefore simply need to optimise linear functions over one subset of constraints at a time.

5.1.1 Inner minimization: $\mathcal{P}_0$ subproblems

To minimize over $\mathbf{z}_0, \hat{\mathbf{z}}_{A,1}$, the variables constrained by $\mathcal{P}_0$, we need to solve:

$$\begin{aligned} [\mathbf{z}_0^*, \hat{\mathbf{z}}_{A,1}^*] = \operatorname*{argmin}_{\mathbf{z}_0, \hat{\mathbf{z}}_{A,1}} \; & -\boldsymbol{\rho}_1^T \hat{\mathbf{z}}_{A,1} \\ \text{s.t} \quad & \mathbf{z}_0 \in \mathcal{C}, \qquad \hat{\mathbf{z}}_{A,1} = W_1 \mathbf{z}_0 + \mathbf{b}_0. \end{aligned} \tag{11}$$

Rewriting the problem as a linear function of $\mathbf{z}_0$ only, problem (11) is simply equivalent to minimising $-\boldsymbol{\rho}_1^T W_1 \mathbf{z}_0$ over $\mathcal{C}$. We assumed that the optimisation of a linear function over $\mathcal{C}$ was efficient. We now give examples of $\mathcal{C}$ where problem (11) can be solved efficiently.

Bounded Input Domain If $\mathcal{C}$ is defined by a set of lower bounds $\mathbf{l}_0$ and upper bounds $\mathbf{u}_0$ (as in the case of ℓ_∞ adversarial examples), optimisation will simply amount to choosing either the lower or upper bound depending on the sign of the linear function. Let us denote the indicator function for condition c by $\mathbb{1}_c$. The optimal solution is:

$$\begin{aligned} \mathbf{x}_0 &= \mathbb{1}_{\boldsymbol{\rho}_1^T W_1 < 0} \cdot \mathbf{l}_0 + \mathbb{1}_{\boldsymbol{\rho}_1^T W_1 \geq 0} \cdot \mathbf{u}_0, \\ \hat{\mathbf{x}}_{A,1} &= W_1 \mathbf{x}_0 + \mathbf{b}_1. \end{aligned} \tag{12}$$

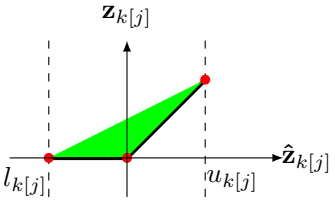


Figure 1: Feasible domain of the convex hull for an ambiguous ReLU. Red circles indicate the vertices of the feasible region.

ℓ_2 Balls If $\mathcal{C}$ is defined by an ℓ_2 ball of radius ϵ around a point $\bar{\mathbf{x}}$ ($\|\mathbf{x}_0 - \bar{\mathbf{x}}\|_2 \leq \epsilon$), optimisation amounts to choosing the point on the boundary of the ball such that the vector from the center to it is opposed to the cost function. Formally, the optimal solution is given by:

$$\begin{aligned} \hat{\mathbf{x}}_{A,1} &= W_1\mathbf{x}_0 + \mathbf{b}_1, \\ \mathbf{x}_0 &= \bar{\mathbf{x}} + (\sqrt{\epsilon}/\|\boldsymbol{\rho}_1\|_2)\boldsymbol{\rho}_1. \end{aligned} \tag{13}$$

5.1.2 Inner minimization: $\mathcal{P}_k$ subproblems

For the variables constrained by subproblem $\mathcal{P}_k$ $(\hat{\mathbf{z}}_{B,k}, \hat{\mathbf{z}}_{A,k+1})$, we need to solve:

$$\begin{aligned} [\hat{\mathbf{z}}^*_{B,k}, \hat{\mathbf{z}}^*_{A,k+1}] = \underset{\hat{\mathbf{z}}_{B,k}, \hat{\mathbf{z}}_{A,k+1}}{\operatorname{argmin}} \quad & \boldsymbol{\rho}_k^T \hat{\mathbf{z}}_{B,k} - \boldsymbol{\rho}_{k+1}^T \hat{\mathbf{z}}_{A,k+1} \\ \text{s.t} \quad & \mathbf{l}_k \leq \hat{\mathbf{z}}_{B,k} \leq \mathbf{u}_k, \\ & \texttt{cvx\_hull}_\sigma\left(\hat{\mathbf{z}}_{B,k}, \mathbf{z}_k, \mathbf{l}_k, \mathbf{u}_k\right), \\ & \hat{\mathbf{z}}_{A,k+1} = W_{k+1}\mathbf{z}_k + \mathbf{b}_{k+1}. \end{aligned} \tag{14}$$

In the case where the activation function σ is the ReLU and `cvx_hull` is given by equation (4), we can find a closed form solution. Using the last equality of the problem, we can rewrite the objective function as $\boldsymbol{\rho}_k^T \hat{\mathbf{z}}_{B,k} - \boldsymbol{\rho}_{k+1}^T W_{k+1}\mathbf{z}_k$. If the ReLU is ambiguous, the shape of the convex hull is represented in Figure 1. If the sign of $\boldsymbol{\rho}_{k+1}^T W_{k+1}$ is negative, optimizing subproblem (14) corresponds to having $\mathbf{z}_k$ at its lower bound $\max\left(\hat{\mathbf{z}}_{B,k}, 0\right)$. If on the contrary the sign is positive, $\mathbf{z}_k$ must be equal to its upper bound $\frac{\mathbf{u}_k}{\mathbf{u}_k - \mathbf{l}_k}\left(\hat{\mathbf{z}}_{B,k} - \mathbf{l}_k\right)$. We can therefore rewrite the subproblem (14) in the case of ambiguous ReLUs as:

$$\begin{aligned} \underset{\hat{\mathbf{z}}_{B,k} \in [\mathbf{l}_k, \mathbf{u}_k]}{\operatorname{argmin}} \quad & \left(\boldsymbol{\rho}_k^T - \left[\boldsymbol{\rho}_{k+1}^T W_{k+1}\right]_+ \frac{\mathbf{u}_k}{\mathbf{u}_k - \mathbf{l}_k}\right) \hat{\mathbf{z}}_{B,k} \\ & - \left[\boldsymbol{\rho}_{k+1}^T W_{k+1}\right]_- \max\left(\hat{\mathbf{z}}_{B,k}, 0\right), \end{aligned} \tag{15}$$

where $[A]_-$ corresponds to the negative values of A ($[A]_- = \min(A, 0)$) and $[A]_+$ corresponds to the positive value of A ($[A]_+ = \max(A, 0)$). The objective function and the constraints all decompose completely over the coordinates, so all problems can be solved independently. For each dimension, the problem is a convex, piecewise linear function, which means that the optimal point will be a vertex. The possible vertices are $(\mathbf{l}_k[i], 0)$, $(0, 0)$, and $(\mathbf{u}_k[i], \mathbf{u}_k[i])$. In order to find the minimum, we can therefore evaluate the objective function at these three points and keep the one with the smallest value.

If for a ReLU we either have $\mathbf{l}_k[i] \geq 0$ or $\mathbf{u}_k[i] \leq 0$, the `cvx_hull` constraint is a simple linear equality constraint. For those coordinates, the problem is analogous to the one solved by equation (12), with the linear function being minimised over the $\hat{\mathbf{z}}_{B,k}$ box bounds being $\boldsymbol{\rho}_k^T \hat{\mathbf{z}}_{B,k}$ in the case of blocking ReLUs or $\left(\boldsymbol{\rho}_k^T - \boldsymbol{\rho}_{k+1}^T W_{k+1}\right) \hat{\mathbf{z}}_{B,k}$ in the case of passing ReLUs.

We have described how to solve problem (14) by simply evaluating linear functions at given points. The matrix operations involved correspond to standard operations of the neural network's layers. Computing $\hat{\mathbf{z}}_{A,k+1} = W_{k+1}\mathbf{z}_k + \mathbf{b}_{k+1}$ is exactly the forward pass of the network and computing $\boldsymbol{\rho}_{k+1}^T W_{k+1}$ is analogous to the backpropagation of gradients. We can therefore take advantage of existing deep learning frameworks to gain access to efficient implementations. When dealing with convolutional layers, we can employ specialised implementations rather than building the equivalent W_k matrix, which would contain a lot of redundancy.

We described here the solving process in the context of ReLU activation functions, but this can be generalised to non piecewise linear activation function. For example, appendix A describes the solution for sigmoid activations.

5.2 PROXIMAL METHOD

We now detail the use of proximal methods on problem (9) as an alternative to supergradient ascent.

5.2.1 Augmented Lagrangian

Applying proximal maximization to the dual function q results in the Augmented Lagrangian Method, also known as the method of multipliers. The derivation of the update steps detailed by Bertsekas and Tsitsiklis (1989). For our problem, this will correspond to alternating between updates to the dual variables $\boldsymbol{\rho}$ and updates to the primal variables $\hat{\mathbf{z}}$, which are given by the following equations (superscript t indicates the value at the t-th iteration):

$$\boldsymbol{\rho}_k^{t+1} = \boldsymbol{\rho}_k^t + \frac{\hat{\mathbf{z}}^t_{B,k} - \hat{\mathbf{z}}^t_{A,k}}{\eta_k}, \tag{16}$$

$$\begin{aligned} \left[\mathbf{z}^t, \hat{\mathbf{z}}^t\right] &= \underset{\mathbf{z}, \hat{\mathbf{z}}}{\operatorname{argmin}} \mathcal{L}\left(\hat{\mathbf{z}}, \boldsymbol{\rho}^t\right) \\ &:= \underset{\mathbf{z}, \hat{\mathbf{z}}}{\operatorname{argmin}} \Bigg[\hat{\mathbf{z}}_{A,n} + \sum_{k=1..n-1} \boldsymbol{\rho}_k^T \left(\hat{\mathbf{z}}_{B,k} - \hat{\mathbf{z}}_{A,k}\right) \\ &\quad + \sum_{k=1..n-1} \frac{1}{2\eta_k} \|\hat{\mathbf{z}}_{B,k} - \hat{\mathbf{z}}_{A,k}\|^2 \Bigg] \\ \text{s.t.} \quad & \mathcal{P}_0(\mathbf{z}_0, \hat{\mathbf{z}}_{A,1}), \\ & \mathcal{P}_k(\hat{\mathbf{z}}_{B,k}, \hat{\mathbf{z}}_{A,k+1}) \quad k \in [1, n-1]. \end{aligned} \tag{17}$$

The term $\mathcal{L}\left(\hat{\mathbf{z}}, \boldsymbol{\rho}\right)$ is the Augmented Lagrangian of problem (8). The additional quadratic term in (17), compared to the objective of $q(\boldsymbol{\rho})$, arises from the proximal terms on $\boldsymbol{\rho}$. It has the advantage of making the problem strongly convex, and hence easier to optimize. Later on, will show that this allows us to derive optimal step-sizes in closed

form. The weight η_k is a hyperparameter of the algorithm. A high value will make the problem more strongly convex and therefore quicker to solve, but it will also limit the ability of the algorithm to perform large updates.

While obtaining the new values of $\boldsymbol{\rho}$ is trivial using equation (16), problem (17) does not have a closed-form solution. We show how to solve it efficiently nonetheless.

5.2.2 Frank-Wolfe Algorithm

Problem (17) can be optimised using the conditional gradient method, also known as the Frank-Wolfe algorithm (Frank and Wolfe, 1956). The advantage it provides is that there is no need to perform a projection step to remain in the feasible domain. Indeed, the different iterates remain in the feasible domain by construction as convex combination of points in the feasible domain. At each time step, we replace the objective by its linear approximation and optimize this linear function over the feasible domain to get an update direction, named conditional gradient. We then take a step in this direction. As the Augmented Lagrangian is smooth over the primal variables, there is no need to take the Frank-Wolfe step for all the network layers at once. We can in fact do it in a block-coordinate fashion, where a block is a network layer, with the goal of speeding up convergence; we refer the reader to appendix C for further details.

Obtaining the conditional gradient requires minimising a linearization of $\mathcal{L}\left(\hat{\mathbf{z}}, \boldsymbol{\rho}\right)$ on the primal variables, restricted to the feasible domain. This computation corresponds exactly to the one we do to perform the inner minimisation of problem (9) over $\mathbf{z}$ and $\hat{\mathbf{z}}$ in order to compute the supergradient (cf. §5.1.1, §5.1.2). To make this equivalence clearer, we point out that the linear coefficients of the primal variables will maintain the same form (with the difference that the dual variables are represented as their closed-form update for the following iteration), as $\nabla_{\hat{\mathbf{z}}_{B,k}} \mathcal{L}\left(\hat{\mathbf{z}}, \boldsymbol{\rho}^t\right) = \boldsymbol{\rho}_k^{t+1}$ and $\nabla_{\hat{\mathbf{z}}_{A,k+1}} \mathcal{L}\left(\hat{\mathbf{z}}, \boldsymbol{\rho}^t\right) = -\boldsymbol{\rho}_k^{t+1}$. The equivalence of conditional gradient and supergradient is not particular to our problem. A more general description can be found in the work of Bach (2015).

The use of a proximal method allows us to employ an optimal step size, whose calculation is detailed in appendix C. This would not be possible with supergradient methods, as we would have no guarantee of improvements. We therefore have no need to choose the step-size. In practice, we still have to choose the strength of the proximal term η_k. Finally, inspired by previous work on accelerating proximal methods (Lin et al., 2017, Salzo and Villa, 2012), we also apply momentum on the dual updates to accelerate convergence; for details see appendix C.

6 EXPERIMENTS

6.1 IMPLEMENTATION DETAILS

One of the benefits of our algorithms (and of the supergradient baseline by Dvijotham et al. (2018)) is that they are easily parallelizable. As explained in § 5.1.1 and § 5.1.2, the crucial part of both our solvers works by applying linear algebra operations that are equivalent to the ones employed during the forward and backward passes of neural networks. To compute the upper and lower bounds for all the neurons of a layer, there is no dependency between the different problems, so we are free to solve them all simultaneously in a batched mode. In complete verification (§6.4), where bounds relative to multiple domains (defined by the set of input and intermediate bounds for the given network activations) we can further batch over the domains, providing a further stream of parallelism. In practice, we take the building blocks provided by deep learning frameworks, and apply them to batches of solutions (one per problem), in the same way that normal training applies them to a batch of samples. This makes it possible for us to leverage the engineering efforts made to enable fast training and evaluation of neural networks, and easily take advantage of GPU accelerations. The implementation used in our experiments is based on Pytorch (Paszke et al., 2017).

6.2 EXPERIMENTAL SETTING

In order to evaluate the performance of the proposed methods, we first perform a comparison in the quality and speed of bounds obtained by different methods in the context of incomplete verification (§6.3). We then assess the effect of the various method-specific speed/accuracy trade-offs within a complete verification procedure (§6.4).

For both sets of experiments, we report results using networks trained with different methodologies on the CIFAR-10 dataset. We compare the following algorithms:

- **WK** uses the method of Wong and Kolter (2018). This is equivalent to a specific choice of $\boldsymbol{\rho}$.
- **DSG+** corresponds to supergradient ascent on dual (2), the method by Dvijotham et al. (2018). We use the Adam (Kingma and Ba, 2015) updates rules and decrease the step size linearly, similarly to the experiments of Dvijotham et al. (2018). We experimented with other step size schedules, like constant step size or $\frac{1}{t}$ schedules, which all performed worse.
- **Dec-DSG+** is a direct application of Theorem 1: it obtains a dual solution for problem (2) via DSG+ and sets $\boldsymbol{\rho} = \boldsymbol{\mu}$ in problem (9) to obtain the final bounds.
- **Supergradient** is the first of the two solvers presented (§5.1). It corresponds to supergradient ascent over problem (9). We use Adam updates as in DSG+.
- **Proximal** is the solver presented in §5.2, performing proximal maximisation on problem (9). We limit the

total number of inner iterations for each problem to 2 (a single iteration is a full pass over the network layers).

- **Gurobi** is our gold standard method, employing the commercial black box solver Gurobi. All the problems are solved to optimality, providing us with the best result that our solver could achieve in terms of accuracy.
- **Gurobi-TL** is the time-limited version of the above, which stops at the first dual simplex iteration for which the total elapsed time exceeded that required by 400 iterations of the proximal method.

We omit **Interval Bound Propagation** (Gowal et al., 2018, Mirman et al., 2018) from the comparisons as on average it performed significantly worse than WK on the networks we used: experimental results are provided in appendix G.1. For all of the methods above, as done in previous work for neural network verification (Bunel et al., 2020, Lu and Kumar, 2020), we compute intermediate bounds (see appendix B) by taking the layer-wise best bounds output by Interval Propagation and WK. This holds true for both the incomplete and the complete verification experiments. Gurobi was run on 4 CPUs for § 6.3 and on 1 for § 6.4 in order to match the setting by Bunel et al. (2020), whereas all the other methods were run on a single GPU. While it may seem to lead to an unfair comparison, the flexibility and parallelism of the methods are a big part of their advantages over off-the-shelf solvers.

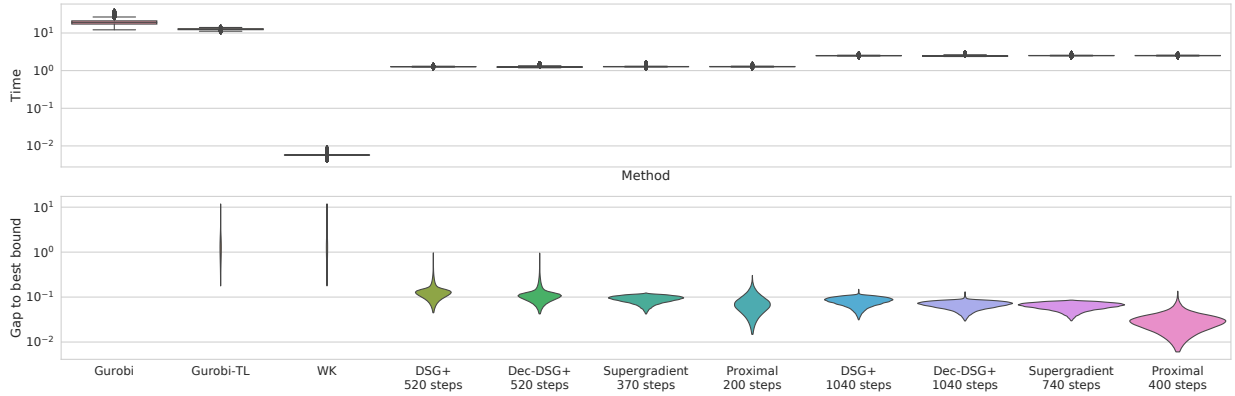
6.3 INCOMPLETE VERIFICATION

We investigate the effectiveness of the various methods for incomplete verification on images of the CIFAR-10 test set. For each image, we compute an upper bound on the robustness margin: the difference between the ground truth logit and all the other logits, under an allowed perturbation ϵ_{verif} in infinity norm of the inputs. If for any class the upper bound on the robustness margin is negative, then we are certain that the network is vulnerable against that adversarial perturbation. We measure the time to compute last layer bounds, and report the optimality gap compared to the best achieved solution.

As network architecture, we employ the small model used by Wong and Kolter (2018) and whose structure is given in appendix G.1.1. We train the network against perturbations of a size up to $\epsilon_{\text{train}} = 8/255$, and test for adversarial vulnerability on $\epsilon_{\text{verif}} = 12/255$. This is done using adversarial training (Madry et al., 2018), based on an attacker using 50 steps of projected gradient descent to obtain the samples. Additional experiments for a network trained using standard stochastic gradient descent and cross entropy, with no robustness objective can be found in appendix G.1. For both supergradient methods (our Supergradient, and DSG+), we decrease the step size linearly from $\alpha = 10^{-2}$ to $\alpha = 10^{-4}$, while for Proximal, we employ momentum coefficient $\mu = 0.3$ and, for all layers, linearly increase the weight of the proximal terms from $\eta = 10^1$ to $\eta = 5 \times 10^2$ (see appendix C).

Figure 2 presents results as a distribution of runtime and optimality gap. WK by Wong and Kolter (2018) performs a single pass over the network per optimization problem, which allows it to be extremely fast, but this comes at the cost of generating looser bounds. At the complete opposite end of the spectrum, Gurobi is extremely slow but provides the best achievable bounds. Dual iterative methods (DSG+, Supergradient, Proximal) allow the user to choose the trade-off between tightness and speed. In order to perform a fair comparison, we fixed the number of iterations for the various methods so that each of them would take the same average time (see Figure 3). This was done by tuning the iteration ratios on a subset of the images for the results in Figures 2, 3. Note that the Lagrangian Decomposition has a higher cost per iteration due to the more expensive computations related to the more complex primal feasible set. The cost of the proximal method is even larger due to the primal-dual updates and the optimal step size computation. For DSG+, Supergradient

Figure 2: Distribution of runtime and gap to optimality on a network adversarially trained with the method by Madry et al. (2018), on CIFAR-10 images. In both cases, lower is better. The width at a given value represents the proportion of problems for which this is the result. Gurobi always return the optimal solution so doesn't appear on the optimality gap, but is always the highest runtime.



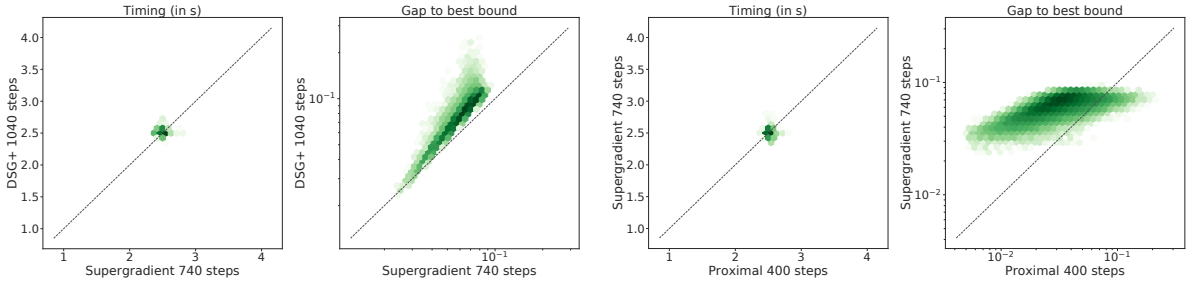


Figure 3: Pointwise comparison for a subset of the methods on the data presented in Figure 2. Each datapoint corresponds to a CIFAR image, darker colour shades mean higher point density. The dotted line corresponds to the equality and in both graphs, lower is better along both axes.

and Proximal, the improved quality of the bounds as compared to the non-iterative method WK shows that there are benefits in actually solving the best relaxation rather than simply evaluating looser bounds. Time-limiting Gurobi at the first iteration which exceeded the cost of 400 Proximal iterations significantly worsens the produced bounds without a comparable cut in runtimes. This is due to the high cost per iteration of the dual simplex algorithm.

By looking at the distributions and at the point-wise comparisons in Figure 3, we can see that Supergradient yields consistently better bounds than DSG+. As both methods employ the Adam update rule (and the same hyperparameters, which are optimal for both), we can conclude that operating on the Lagrangian Decomposition dual (1) produces better speed-accuracy trade-offs compared to the dual (2) by Dvijotham et al. (2018). This is in line with the expectations from Theorem 1. Furthermore, while a direct application of the Theorem (Dec-DSG+) does improve on the DSG+ bounds, operating on the Decomposition dual (1) is empirically more effective (see pointwise comparisons in appendix G.1). Finally, on average, the proximal algorithm yields better bounds than those returned by Supergradient, further improving on the DSG+ baseline. In particular, we stress that the support of optimality gap distribution is larger for Proximal, with a heavier tail towards better bounds.

6.4 COMPLETE VERIFICATION

We present results for complete verification. In this setting, our goal is to verify whether a network is robust to ℓ_∞ norm perturbations of radius ϵ_{verif}. In order to do so, we search for a counter-example (an input point for which the output of the network is not the correct class) by minimizing the difference between the ground truth logit and a randomly chosen logit of images of the CIFAR-10 test set. If the minimum is positive, we have not succeeded in finding a counter-example, and the network is robust. In contrast to the previous section, we now seek to solve a nonconvex problem like (1) (where another layer representing the aforementioned difference has been added at the end) directly, rather than an approximation.

In order to perform the minimization exactly, we employ BaSBR, a Branch and Bound algorithm by Bunel et al. (2020). In short, Branch and Bound divides the verification domain into a number of smaller problems, for which it repeatedly computes upper and lower bounds. At every iteration, BaSBR picks the sub-problem with the lowest lower bound and creates two new sub-problems by fixing the most "impactful" ReLU to be passing or blocking. The impact of a ReLU is determined by estimating the change in the sub-problem's output lower bound caused by making the ReLU non-ambiguous. Sub-problems which cannot contain the global lower bound

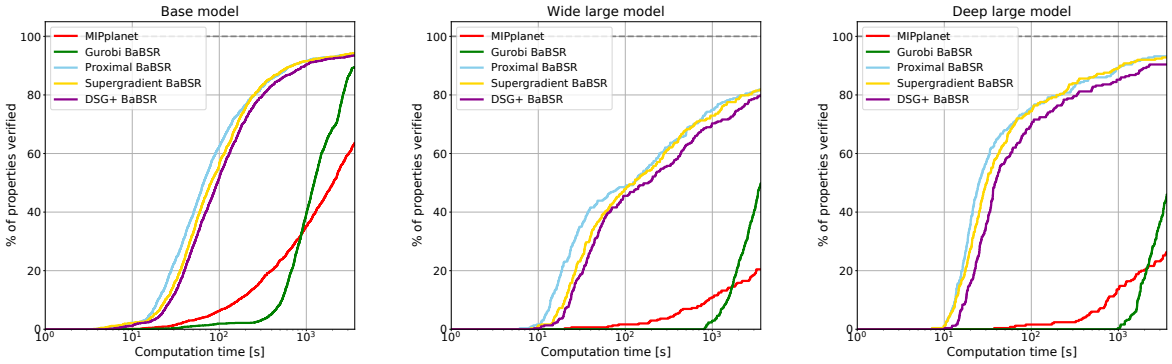


Figure 4: Cactus plots for the base, wide and deep models. For each, we compare the bounding methods and complete verification algorithms by plotting the percentage of solved properties as a function of runtime.

Table 1: For base, wide and deep models, we compare average solving time, average number of solved sub-problems and the percentage of timed out properties. The best performing iterative method is highlighted in bold.

	Base			Wide			Deep		
Method	time(s)	sub-problems	%Timeout	time(s)	sub-problems	%Timeout	time(s)	sub-problems	%Timeout
GUROBI BABSR	1588.651	1342.777	10.56	2912.246	630.864	50.66	3007.237	299.913	54.00
MIPPLANET	2035.565		36.37	3118.593		80.00	2997.115		73.60
DSG+ BABSR	426.554	5312.674	6.54	1048.594	4518.116	20.00	509.484	1992.345	9.60
SUPERGRADIENT BABSR	377.104	5079.682	**5.76**	920.351	4069.894	18.00	390.367	1783.577	7.2
PROXIMAL BABSR	**368.876**	**4569.414**	5.82	**891.163**	**3343.601**	**18.00**	**384.984**	**1655.519**	**6.8**

are progressively discarded.

The computational bottleneck of Branch and Bound is the lower bounds computation, for which we will employ a subset of the methods in section 6.2. Specifically, we want to compare the performance of DSG+, Supergradient and Proximal with Gurobi (as employed in Bunel et al. (2020)). For this purpose, we run the various Branch and Bound implementations on the dataset employed by Lu and Kumar (2020) to test their novel Branch and Bound splitting strategy. The dataset picks a non-correct class and a perturbation radius ϵ_{verif} for a subset of the CIFAR-10 test images, and runs Branch and Bound on three different convolutional network architectures of varying size: a "base" network, a "wide" network, and a "deep" network. Further details on the dataset and architectures are provided in appendix G.2.

We compare the Branch and Bound implementations with MIPplanet to provide an additional verification baseline. MIPplanet computes the global lower bound by using Gurobi to solve the Mixed-Integer Linear problem arising from the Planet relaxation (Bunel et al., 2020, Ehlers, 2017). As explained in section 6.1, due to the highly parallelizable nature of the dual iterative algorithms, we are able to compute lower bounds for multiple sub-problems at once for DSG+, Supergradient and Proximal, whereas the domains are solved sequentially for Gurobi. The number of simultaneously solved sub-problems is 300 for the base network, and 200 for the wide and deep networks. Intermediate bound computations (see §6.2) are performed in parallel as well, with smaller batch sizes to account for the larger width of intermediate layers (over which we batch as well). For both supergradient methods (our Supergradient, and DSG+), we decrease the step size linearly from $\alpha = 10^{-3}$ to $\alpha = 10^{-4}$, while for Proximal, we do not employ momentum and keep the weight of the proximal terms fixed to $\eta = 10^2$ for all layers throughout the iterations. As in the previous section, the number of iterations for the bounding algorithms are tuned to employ roughly the same time: we use 100 iterations for Proximal, 160 for Supergradient, and 260 for DSG+. For all three, the dual variables are initialized from the dual solution of the lower bound computation of the parent node in the Branch and Bound tree. We time-limit the experiments at one hour.

Consistently with the incomplete verification results in the last section, Figure 4 and Table 1 show that the Supergradient overperforms DSG+, confirming the benefits of our Lagrangian Decomposition approach. Better bounds decrease the number of sub-problems that Branch and Bound needs to solve, reducing the overall verification time. Furthermore, Proximal provides an additional increase in performance over DSG+, which is visible especially over the properties that are easier to verify. The gap between competing bounding methods increases with the size of the employed network, both in terms of layer width and network depth: at least an additional 2% of the properties times out when using DSG+ on the larger networks. A more detailed experimental analysis of the base model data is presented in appendix G.2.

7 DISCUSSION

We have presented a novel dual approach to compute bounds over the activation of neural networks based on Lagrangian Decomposition. It provides significant benefits compared to off-the-shelf solvers and improves on both looser relaxations and on a previous method based on Lagrangian relaxation. As future work, it remains to investigate whether better complete verification results can be obtained by combining our supergradient and proximal methods. Furthermore, it would be interesting to see if similar derivations could be used to make tighter but more expensive relaxations computationally feasible. Improving the quality of bounds may allow the training of robust networks to avoid over-regularisation.

Acknowledgments

ADP was supported by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems, grant EP/L015987/1. RB wishes to thank Leonard Berrada for helpful discussions on the convex hull of sigmoid activation functions.

References

Anderson, R., Huchette, J., Tjandraatmadja, C., and Vielma, J. P. (2019). Strong mixed-integer programming formulations for trained neural networks. *Conference on Integer Programming and Combinatorial Optimization*.

Athalye, A., Carlini, N., and Wagner, D. A. (2018). Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples. *International Conference on Machine Learning*.

Bach, F. (2015). Duality between subgradient and condi-

tional gradient methods. *SIAM Journal on Optimization*.

Bertsekas, D. P. and Tsitsiklis, J. N. (1989). *Parallel and distributed computation: numerical methods*. Prentice hall Englewood Cliffs, NJ.

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

Bunel, R., Turkaslan, I., Torr, P. H., Kohli, P., and Kumar, M. P. (2018). A unified view of piecewise linear neural network verification. *Neural Information Processing Systems*.

Cheng, C.-H., Nührenberg, G., and Ruess, H. (2017). Maximum resilience of artificial neural networks. *Automated Technology for Verification and Analysis*.

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

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

Frank, M. and Wolfe, P. (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterly*.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. *International Conference on Learning Representations*.

Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Mann, T., and Kohli, P. (2018). On the effectiveness of interval bound propagation for training verifiably robust models. *Workshop on Security in Machine Learning, NeurIPS*.

Guignard, M. and Kim, S. (1987). Lagrangean decomposition: A model yielding stronger lagrangean bounds. *Mathematical programming*.

Gurobi Optimization, L. (2020). Gurobi optimizer reference manual.

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

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Lin, H., Mairal, J., and Harchaoui, Z. (2017). Catalyst acceleration for first-order convex optimization: From theory to practice. *Journal of Machine Learning Research*.

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

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

Mirman, M., Gehr, T., and Vechev, M. (2018). Differentiable abstract interpretation for provably robust neural networks. *International Conference on Machine Learning*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. *NIPS Autodiff Workshop*.

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

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

Salzo, S. and Villa, S. (2012). Inexact and accelerated proximal point algorithms. *Journal of Convex Analysis*, 19:1167–1192.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. *International Conference on Learning Representations*.

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

Uesato, J., O'Donoghue, B., Oord, A. v. d., and Kohli, P. (2018). Adversarial risk and the dangers of evaluating against weak attacks. *International Conference on Machine Learning*.

Weng, T.-W., Zhang, H., Chen, H., Song, Z., Hsieh, C.-J., Boning, D., Dhillon, I. S., and Daniel, L. (2018). Towards fast computation of certified robustness for relu networks. *International Conference on Machine Learning*.

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

Xiang, W., Tran, H.-D., and Johnson, T. T. (2017). Output reachable set estimation and verification for multi-layer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.