Partition-Based Formulations for Mixed-Integer Optimization of Trained ReLU Neural Networks

Calvin Tsay, Jan Kronqvist, Alexander Thebelt, Ruth Misener
Department of Computing, Imperial College London
South Kensington, SW7 2AZ, United Kingdom
{c.tsay, j.kronqvist, alexander.thebelt18, r.misener}@imperial.ac.uk

***Abstract*—This paper introduces a class of mixed-integer formulations for trained ReLU neural networks. The approach balances model size and tightness by partitioning node inputs into a number of groups and forming the convex hull over the partitions via disjunctive programming. At one extreme, one partition per input recovers the convex hull of a node, i.e., the tightest possible formulation for each node. For fewer partitions, we develop smaller relaxations that approximate the convex hull, and show that they outperform existing formulations. Specifically, we propose strategies for partitioning variables based on theoretical motivations and validate these strategies using extensive computational experiments. Furthermore, the proposed scheme complements known algorithmic approaches, e.g., optimization-based bound tightening captures dependencies within a partition.**

I. Introduction

Many applications use mixed-integer linear programming (MILP) to optimize over trained feed-forward ReLU neural networks [1]–[6]. A MILP encoding of a ReLU-NN enables network properties to be rigorously analyzed, e.g., verifying robustness of an output (often classification) within a restricted input domain [7]. MILP encodings of ReLU-NNS have also been used to determine robust perturbation bounds [8], compress NNs [9], count linear regions [10], and find adversarial examples [11]. The so-called *big-M* formulation is the main approach for encoding NNs as MILPs and is used in all above references. Optimizing the resulting MILPs remains challenging for large networks, even with state-of-the-art software.

Effectively solving a MILP hinges on the strength of its continuous relaxation [12]; weak relaxations can render MILP problems computationally intractable. For NNs, Anderson et al. [13] showed that the big-M formulation is not tight and presented formulations for the convex hull (i.e., the tightest possible formulation) of individual nodes. However, the convex hull formulations require either an exponential (with respect to node inputs) number of constraints or many additional/auxiliary variables. So, despite its weaker continuous relaxation, the big-M formulation can be computationally advantageous owing to its smaller size.

Given these challenges, we present a novel class of MILP formulations for ReLU-NNs. The formulations are hierarchical: their relaxations start at a big-M equivalent and converge to the convex hull. Intermediate formulations can closely approximate the convex hull with many fewer variables/constraints. The formulations are constructed by viewing each ReLU node as a two-part disjunction. Kronqvist et al. [14] proposed hierarchical relaxations for general disjunctive programs. This work uses a similar hierarchy to construct strong and efficient MILP formulations for ReLU-NNs. Specifically, we partition the inputs of each node into groups and formulate the convex hull over the resulting groups. With fewer groups than inputs, this approach results in MILPs that are smaller than convex-hull formulations, yet have stronger relaxations than big-M.

Three optimization tasks evaluate the new formulations: *optimal adversarial examples*, *robust verification*, and ℓ_1-*minimally distorted adversaries*. Extensive computation shows that our formulations outperform the standard big-M approach with 25% more problems solved within a 1h time limit (average 2.2X speedups for solved problems).

Related work. Techniques for obtaining strong relaxations of ReLU-NNs include linear programming [15]–[17], semidefinite programming [18], [19], Lagrangian decomposition [20], combined relaxations [21], and relaxations over multiple nodes [22]. These relaxation techniques do not exactly represent ReLU-NNs, but rather derive valid bounds for the network in general. These techniques might fail to verify some properties, due to their non-exactness, but they can be much faster than MILP-based techniques.

Strong MILP encoding of ReLU-NNs was also studied in [13]. Our approach is fundamentally different, as it constructs computationally cheaper formulations that approximate the convex hull, and we start by deriving a stronger relaxation instead of strengthening the relaxation via cutting planes. Furthermore, our formulation enables input node dependencies to easily be incorporated.

Contributions of this paper. We present a new class of strong, yet compact, MILP formulations for feed-forward ReLU-NNs. Section III-A observes how, in conjunction with optimization-based bound tightening, partitioning input variables can efficiently incorporate dependencies into MILP formulations. Section III-B builds on the convex disjunctive programming relaxations in [14] to prove the hierarchical nature of the proposed formulations for ReLU-NNs, with relaxations spanning between big-M and convex-hull formulations. Sections III-C–III-D show that formulation tightness depends on the specific choice of variable partitions, and we present efficient partitioning strategies based on both theoretical and computational motivations. The advantages of the new formulations are demonstrated via extensive computational experiments in Section IV.

II. Background

A. Feed-forward Neural Networks

A feed-forward neural network (NN) is a directed acyclic graph with nodes structured into a number of k layers. Layer k receives the outputs of nodes in the preceding layer $k-1$ as its inputs (layer 0 represents inputs to the NN). Each node in a layer computes a weighted sum of its inputs (known as the *preactivation*), and applies an activation function. This work considers the ReLU activation function:

$$y = \max(0, \boldsymbol{w}^T \boldsymbol{x} + b) \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^\eta$ and $y \in [0, \infty)$ are, respectively, the inputs and output of a node ($\boldsymbol{w}^T\boldsymbol{x} + b$ is the preactivation). Parameters $\boldsymbol{w} \in \mathbb{R}^\eta$ and $b \in \mathbb{R}$ are its weights and bias.

B. ReLU Optimization Formulations

In contrast to the training of NNs (where parameters $\boldsymbol{w}$ and b are optimized), optimization over a NN seeks extreme cases for a *trained* model. Therefore, model parameters $(\boldsymbol{w}, b)$ are fixed, and the inputs/outputs of nodes in the network $(\boldsymbol{x}, y)$ are optimization variables instead.

Big-M Formulation. The ReLU function (1) is commonly modeled [2], [8]:

$$y \geq (\boldsymbol{w}^T\boldsymbol{x} + b) \tag{2}$$

$$y \leq (\boldsymbol{w}^T\boldsymbol{x} + b) - (1-\sigma)LB^0 \tag{3}$$

$$y \leq \sigma\, UB^0 \tag{4}$$

where $\sigma \in \{0, 1\}$ is a binary variable corresponding to the on-off state of the neuron. The formulation requires the bounds (big-M coefficients) LB^0, $UB^0 \in \mathbb{R}$, which should be as tight as possible, such that $(\boldsymbol{w}^T\boldsymbol{x} + b) \in [LB^0, UB^0]$.

Disjunctive Programming [23]. We observe that (1) can be alternatively modeled as a disjunctive program:

$$\begin{bmatrix} y = 0 \\ \boldsymbol{w}^T\boldsymbol{x} + b \leq 0 \end{bmatrix} \vee \begin{bmatrix} y = \boldsymbol{w}^T\boldsymbol{x} + b \\ \boldsymbol{w}^T\boldsymbol{x} + b \geq 0 \end{bmatrix} \tag{5}$$

Note that the second equation is not necessary to model y, but in general results in a tighter formulation for $\boldsymbol{x}$. The extended formulation for disjunctive programs introduces auxiliary variables for each disjunction. Defining $z := \boldsymbol{w}^T\boldsymbol{x}$, $z^a \in \mathbb{R}$ and $z^b \in \mathbb{R}$ can be introduced to model (5):

$$\boldsymbol{w}^T\boldsymbol{x} = z^a + z^b \tag{6}$$

$$z^a + \sigma b \leq 0 \tag{7}$$

$$z^b + (1-\sigma)b \geq 0 \tag{8}$$

$$y = z^b + (1-\sigma)b \tag{9}$$

$$\sigma LB^a \leq z^a \leq \sigma\, UB^a \tag{10}$$

$$(1-\sigma)LB^b \leq z^b \leq (1-\sigma)\, UB^b \tag{11}$$

where the summation in (6) can be used to eliminate either z^a or z^b. Therefore, in practice, only one auxiliary variable is introduced by the formulation (6)–(11).

Relaxation strength. MILP is often solved with branch-and-bound, a strategy that bounds the objective function between a feasible point and its optimal relaxation. The integral search space is explored by "branching" until the gap between bounds reaches a desired value. A *tighter*, or stronger, relaxation can reduce this search tree considerably.

III. Disaggregated Disjunctions: Between Big-M and the Convex Hull

Our proposed formulations split the sum $z = \boldsymbol{w}^T\boldsymbol{x}$ into partitions: we will show these formulations are tighter than (6)–(11). In particular, we partition the set $\{1, ..., \eta\}$ into subsets $\mathbb{S}_1 \cup \mathbb{S}_2 \cup ... \cup \mathbb{S}_N = \{1, ..., \eta\}$; $\mathbb{S}_n \cap \mathbb{S}_{n'} = \emptyset\ \forall n \neq n'$. An auxiliary variable is then introduced for each partition, i.e., $z_n = \sum_{\mathbb{S}_n} w_i x_i$. Replacing $z = \boldsymbol{w}^T \boldsymbol{x}$ with $\sum_n z_n, n = 1, ..., N$, the disjunction (5) becomes:

$$\left[\begin{array}{c} y = 0 \\ \sum_{n=1}^N z_n + b \leq 0 \end{array} \right] \vee \left[\begin{array}{c} y = \sum_{n=1}^N z_n + b \\ y \leq 0 \end{array} \right] \quad (12)$$

The extended formulation then introduces auxiliary variables z_n^a and z_n^b for each z_n:

$$\sum_{i \in \mathbb{S}_n} w_i x_i = z_n^a + z_n^b \quad (13)$$

$$\sum_n z_n^a + \sigma b \leq 0 \quad (14)$$

$$\sum_n z_n^b + (1 - \sigma) b \geq 0 \quad (15)$$

$$y = \sum_n z_n^b + (1 - \sigma) b \quad (16)$$

$$\sigma LB_n^a \leq z_n^a \leq \sigma \, UB_n^a, \forall n = 1, ..., N \quad (17)$$

$$(1 - \sigma) LB_n^b \leq z_n^b \leq (1 - \sigma) \, UB_n^b, \forall n = 1, ..., N \quad (18)$$

where again $\sigma \in \{0, 1\}$. We observe that (13)–(18) *exactly represents the ReLU node* described by (12): substituting $z_n = \sum_{\mathbb{S}_n} w_i x_i$ in the extended convex hull formulation [23] of disjunction (12), directly gives (13)–(18).

Eliminating z_n^a via (13) results in our proposed formulation:

$$\sum_n \left(\sum_{i \in \mathbb{S}_n} w_i x_i - z_n^b \right) + \sigma b \leq 0 \quad (19)$$

$$\sum_n z_n^b + (1 - \sigma) b \geq 0 \quad (20)$$

$$y = \sum_n z_n^b + (1 - \sigma) b, \quad \sigma \in \{0, 1\} \quad (21)$$

$$\sigma LB_n^a \leq \sum_{i \in \mathbb{S}_n} w_i x_i - z_n^b \leq \sigma \, UB_n^a, \forall n = 1, ..., N \quad (22)$$

$$(1 - \sigma) LB_n^b \leq z_n^b \leq (1 - \sigma) \, UB_n^b, \forall n = 1, ..., N \quad (23)$$

Note that domains $[LB_n^a, UB_n^a]$ and $[LB_n^b, UB_n^b]$ may not be equivalent, owing to the inequality constraints in (12).

A. Obtaining and Tightening Bounds

The big-M formulation (2)–(4) requires valid bounds $(\boldsymbol{w}^T \boldsymbol{x} + b) \in [LB^0, UB^0]$. Given bounds for each input variable, $x_i \in [\underline{x}_i, \bar{x}_i]$, interval arithmetic gives valid bounds:

$$LB^0 = \sum_i (\underline{x}_i \max(0, w_i) + \bar{x}_i \min(0, w_i)) + b \quad (24)$$

$$UB^0 = \sum_i (\bar{x}_i \max(0, w_i) + \underline{x}_i \min(0, w_i)) + b \quad (25)$$

But (24)–(25) do not provide the tightest valid bounds in general, as dependencies between the input nodes are ignored. Propagating the resulting over-approximated bounds from layer to layer leads to increasingly large over-approximations, i.e., propagating weak bounds through multiple layers results in a significantly weakened model. Note that these bounds remain in the proposed formulation (19)–(23) in the form of bounds on both the output y and the original variables $\boldsymbol{x}$ (i.e., outputs of the previous layer).

Optimization-Based Bound Tightening (OBBT) or *progressive bounds tightening* [4], tightens variable bounds and constraints [24]. For example, solving the optimization problem with the objective set to minimize/maximize $(\boldsymbol{w}^T \boldsymbol{x} + b)$ gives bounds: $\min(\boldsymbol{w}^T \boldsymbol{x} + b) \leq \boldsymbol{w}^T \boldsymbol{x} + b \leq \max(\boldsymbol{w}^T \boldsymbol{x} + b)$. To simplify these problems, OBBT can be performed using the *relaxed* model (i.e., $\sigma \in [0, 1]$ rather than $\sigma \in \{0, 1\}$), resulting in a linear program (LP). In contrast to the bounds from (24)–(25), bounds from OBBT incorporate variable dependencies. We apply OBBT by solving one LP per bound.

The partitioned formulation (19)–(23) requires bounds such that $z_n^a \in \sigma[LB_n^a, UB_n^a]$, $z_n^b \in (1 - \sigma)[LB_n^b, UB_n^b]$. In other words, $[LB_n^a, UB_n^a]$ is a valid domain for z_n^a when the node is inactive ($\sigma = 1$) and vice versa. These bounds can also be obtained via OBBT: $\min(\sum_{\mathbb{S}_n} w_i x_i) \leq z_n^a \leq \max(\sum_{\mathbb{S}_n} w_i x_i); \boldsymbol{w}^T \boldsymbol{x} + b \leq 0$. The constraint on the right-hand side of disjunction (12) can be similarly enforced in OBBT problems for z_n^b. In our framework, OBBT additionally captures dependencies among each partition $\mathbb{S}_n$. Specifically, we observe that the partitioned OBBT problems effectively form the convex hull over a given polyhedron of the input variables, in contrast to the convex hull formulation, which only considers the box domain defined by the min/max of each input node [13].

Since $\sum_{\mathbb{S}_n} w_i x_i = z_n^a + z_n^b$ and $z_n^a z_n^b = 0$, the bounds $[\min(\sum_{\mathbb{S}_n} w_i x_i), \max(\sum_{\mathbb{S}_n} w_i x_i)]$ are valid for both z_n^a and z_n^b. These bounds can be from (24)–(25), or by solving two OBBT problems for each auxiliary variable z_n ($2N$ LPs total). This simplification uses equivalent bounds for z_n^a and z_n^b, but tighter bounds can potentially be found by performing OBBT for z_n^a and z_n^b separately ($4N$ LPs total).

Figure 1 compares tightness of the proposed formulation with bounds from interval arithmetic (top row) vs OBBT (bottom row). The true model outputs (blue), and minimum (orange) and maximum (green) outputs of the relaxed model are shown over the domain of inputs. The

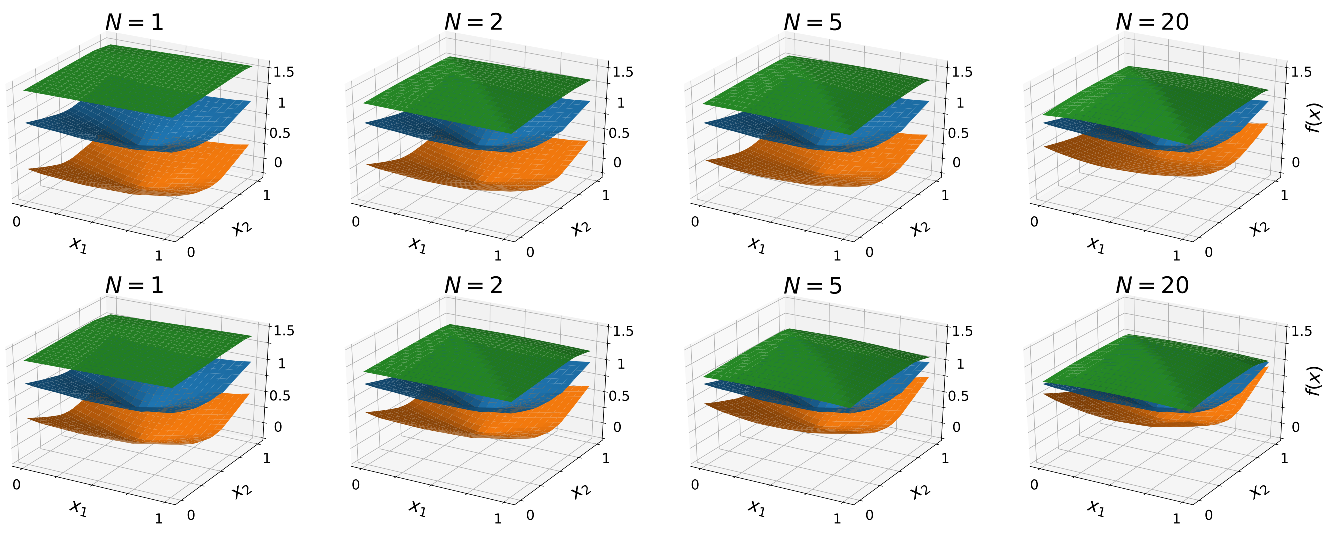


Fig. 1. Hierarchical relaxations from $N = 1$ (equiv. big-M) to $N = 20$ (convex hull of each node over a box domain) for a two-input (x_1, x_2) NN trained on scaled Ackley function, with output $f(\boldsymbol{x})$. Top row: z_n^b bounds obtained using interval arithmetic; Bottom row: z_n^b bounds obtained by optimization-based bound tightening. The partitions are formed using the *equal size* strategy.

NNs are trained on the scaled 2-D Ackley function and comprise two inputs, three hidden layers with 20 nodes each, and a single output. As expected, OBBT greatly improves relaxation tightness.

B. Tightness of the Proposed Formulation

Proposition 1. *Formulation* (19)–(23) *has the equivalent non-lifted (i.e., without auxiliary variables) formulation:*

$$y \leq \sum_{i \in \mathcal{I}_j} w_i x_i + \sigma(b + \sum_{i \in \mathcal{I} \setminus \mathcal{I}_j} UB_i) + (\sigma - 1)(\sum_{i \in \mathcal{I}_j} LB_i), \forall j = 1, ..., 2^N \quad (26)$$

$$y \geq \boldsymbol{w}^T \boldsymbol{x} + b \quad (27)$$

$$y \leq \sigma UB^0 \quad (28)$$

where UB_i and LB_i denote, respectively, the upper and lower bounds of $w_i x_i$. The set $\mathcal{I}$ denotes the input indices $\{1, ..., \eta\}$, and the subset $\mathcal{I}_j$ contains the union of the j-th combination of partitions $\{\mathbb{S}_1, ..., \mathbb{S}_N\}$.

Proof. Formulation (19)–(23) introduces N auxiliary variables $z_n^b, n = 1, ..., N$, which can be projected out using Fourier-Motzkin elimination. The constraint (26) is thus enumerated for all combinations of size $1, .., N$ from the partitions, resulting in combinations $\mathcal{I}_1, ..., \mathcal{I}_J, J = 2^N$. □

Proposition 2. *Formulation* (19)–(23) *for the case of $N = 1$ is equivalent to the big-M formulation* (2)–(4).

Proof. When $N = 1$, it follows that $\mathbb{S}_1 = \{1, .., \eta\}$. Therefore, $z_1 = \sum_{i=1}^{\eta} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$, and $\sum_n z_n = z_1 = z$. Conversely, big-M can be seen as the convex hull over a single aggregated "variable," $z = \boldsymbol{w}^T \boldsymbol{x}$. □

Proposition 3. *Formulation* (19)–(23) *represents the convex hull of* (1) *for the case of $N = \eta$.*

Proof. When $N = \eta$, it follows that $\mathbb{S}_n = \{n\}, \forall n = 1, .., \eta$, and, consequently, $z_n = w_n x_n$. The auxiliary variables can be expressed in terms of x_n, rather of z_n (i.e., $z_n^a = w_n x_n^a$ and $z_n^b = w_n x_n^b$):

$$x_n = x_n^a + x_n^b \quad (29)$$

$$\boldsymbol{w}^T \boldsymbol{x}^a + \sigma b \leq 0 \quad (30)$$

$$\boldsymbol{w}^T \boldsymbol{x}^b + (1 - \sigma) b \geq 0 \quad (31)$$

$$y = \boldsymbol{w}^T \boldsymbol{x}^b + (1 - \sigma) b \quad (32)$$

$$\sigma \frac{LB_n^a}{w_n} \leq x_n^a \leq \sigma \frac{UB_n^a}{w_n}, \forall n = 1, ..., \eta \quad (33)$$

$$(1 - \sigma) \frac{LB_n^b}{w_n} \leq x_n^b \leq (1 - \sigma) \frac{UB_n^b}{w_n}, \forall n = 1, ..., \eta \quad (34)$$

This formulation with a copy of each input represents the convex hull of the neuron, e.g., see [13]; however, the overall model tightness still strongly depends on the bounds used for $x_n, n = 1, ..., \eta$. □

Proposition 4. *A formulation with N partitions is strictly tighter than any formulation with $(N - 1)$ partitions that is derived by combining two partitions in the former.*

Proof. When combining two partitions, i.e., $\mathbb{S}'_{N-1} := \mathbb{S}_{N-1} \cup \mathbb{S}_N$, constraints in (26) where $\mathbb{S}'_{N-1} \subseteq \mathcal{I}_j$ are also obtained by $\{\mathbb{S}_{N-1}, \mathbb{S}_N\} \subseteq \mathcal{I}_j$. In contrast, those obtained by $\mathbb{S}_{N-1} \vee \mathbb{S}_N \subseteq \mathcal{I}_j$ cannot be modeled by $\mathbb{S}'_{N-1}$. Since each constraint in (26) is facet-defining [13], omissions result in a less tight formulation. □

Remark. *The convex hull can be formulated with η auxiliary variables* (29)–(34)*, or 2^η constraints* (26)*. While these formulations have tighter relaxations, they can perform worse than big-M due to having more difficult branch-and-bound subproblems. Our proposed formulation balances this tradeoff by introducing a hierarchy of relaxations with increasing tightness and size. The convex hull is created over partitions $z_n, n = 1, ..., N$, rather than the input variables $x_n, n = 1, ..., \eta$. Therefore, only N auxiliary variables or 2^N constraints are introduced, with $N \leq \eta$.*

Figure 1 shows a hierarchy of increasingly tight formulations from $N = 1$ (equiv. big-M) to $N = 20$ (convex hull of each node over a box input domain). The intermediate formulations approximate the convex-hull ($N = 20$) formulation well, but need fewer auxiliary variables/constraints.

C. Effect of Input Partitioning Choice

Formulation (19)–(23) creates the convex hull over $z_n = \sum_{\mathbb{S}_n} w_i x_i, n = 1, ..., N$. Therefore, N dictates model size, but the choice of subsets, $\mathbb{S}_1, ..., \mathbb{S}_N$, can strongly impact its relaxation. By Proposition 4, (19)–(23) can in fact give multiple hierarchies of formulations. While the hierarchies eventually converge to the convex hull, we are especially interested in those with tight relaxations for small N.

Bounds and Bounds Tightening. Bounds on the partitions play a key role in the proposed formulation. For example, consider when a node is inactive: $\sigma = 1$, $z^b_n = 0$, and (22) gives the bounds $\sigma LB^a_n \leq \sum_{\mathbb{S}_n} w_i x_i \leq \sigma UB^a_n$. Intuitively, the proposed formulation represents the convex hull over the auxiliary variables, $z_n = \sum_{\mathbb{S}_n} w_i x_i$, and their bounds play a key role in model tightness. We hypothesize these bounds are most effective when partitions $\mathbb{S}_n$ are selected such that $w_i x_i, \forall i \in \mathbb{S}_n$ are of similar orders of magnitude.

Consider for instance the case of $\boldsymbol{w} = [1, 1, 100, 100]$ and $x_i \in [0, 1], i = 1, ..., 4$. As all weights are positive, interval arithmetic gives $0 \leq \sum x_i w_i \leq \sum \bar{x}_i w_i$.

With two partitions, the choices of $\mathbb{S}_1 = \{1, 2\}$ vs $\mathbb{S}_1 = \{1, 3\}$ give:

$$\begin{bmatrix} x_1 + x_2 \leq \sigma 2 \\ x_3 + x_4 \leq \sigma 2 \end{bmatrix} \text{or} \begin{bmatrix} x_1 + 100x_3 \leq \sigma 101 \\ x_2 + 100x_4 \leq \sigma 101 \end{bmatrix} \quad (35)$$

where σ is a binary variable. The constraints on the right closely approximate the η-partition (i.e., convex hull) bounds: $x_3 \leq \sigma$ and $x_4 \leq \sigma$. But x_1 and x_2 are relatively unaffected by a perturbation $\sigma = 1 - \delta$ (when σ is relaxed). Whereas the formulation on the left constrains the four variables equivalently. If the behavior of the node is dominated by a few inputs, the formulation on the right is strong, as it approximates the convex hull over those inputs ($z_1 \approx x_3$ and $z_2 \approx x_4$ in this case). For the practical case of $N << \eta$, there are likely fewer partitions than dominating variables.

The size of the partitions can also be selected to be uneven:

$$\begin{bmatrix} x_1 \leq \sigma 1 \\ x_2 + 100x_3 + 100x_4 \leq \sigma 201 \end{bmatrix} \text{or} \begin{bmatrix} x_3 \leq \sigma 1 \\ x_1 + x_2 + 100x_4 \leq \sigma 102 \end{bmatrix} \quad (36)$$

Similar tradeoffs are seen here: the first formulation provides the tightest bound for x_1, but x_2 is effectively unconstrained and x_3, x_4 approach the "equal treatment" constraint above. The second formulation provides the tightest bound for x_3, and a tight bound for x_4, but x_1, x_2 are effectively unbounded for fractional σ.

The above analyses also apply to the case of OBBT. For the above example, solving a relaxed model for $\max(x_1 + x_2)$ obtains a bound that affects the two variables equivalently, while the same procedure for $\max(x_1 + 100x_3)$ obtains a bound that is much stronger for x_3 than for x_1. Similarly, $\max(x_1 + x_2)$ captures dependency between the two variables, while $\max(x_1 + 100x_3) \approx \max(100x_3)$.

Relaxation Tightness. The partitions (and their bounds) also directly affect the tightness of the relaxation for the output variable y. The equivalent non-lifted realization (26) reveals the tightness of the above simple example. The partitions $\mathbb{S}_1 = \{1, 3\}, \mathbb{S}_2 = \{2, 4\}$ result in the constraints:

$$y \leq x_1 + 100x_3 + \sigma(b + 101) \quad (37)$$

$$y \leq x_2 + 100x_4 + \sigma(b + 101) \quad (38)$$

Note that combinations $\mathcal{I}_j = \emptyset$ and $\mathcal{I}_j = \{1, 2, 3, 4\}$ in (26) are not analyzed here, as they correspond to the big-M/1-partition model and are unaffected by choice of partitions. The 4-partition model is the tightest formulation and (in addition to all possible 2-partition constraints) includes:

$$y \leq x_i + \sigma(b + 201), i = 1, 2 \tag{39}$$

$$y \leq 100x_i + \sigma(b + 102), i = 3, 4 \tag{40}$$

The 2-partition (37)–(38) closely approximates two of the 4-partition (i.e., convex hull) constraints (40). Analogous to the tradeoffs in terms of bound tightness, we see that this formulation essentially neglects the dependence of y on x_1, x_2 and instead creates the convex hull over $z_1 = x_1 + 100x_3 \approx x_3$ and $z_2 \approx x_4$. For this simple example, the behavior of y is dominated by x_3 and x_4, and this turns out to be a relatively strong formulation. However, when $N << \eta$, we expect neglecting the dependence of y on some input variables to weaken the model.

The alternative partitions $\mathbb{S}_1 = \{1, 2\}, \mathbb{S}_2 = \{3, 4\}$ give:

$$y \leq x_1 + x_2 + \sigma(b + 200) \tag{41}$$

$$y \leq 100x_3 + 100x_4 + \sigma(b + 2) \tag{42}$$

This formulation handles the four variables similarly and creates the convex hull in two new directions: $z_1 = x_1 + x_2$ and $z_2 = 100(x_3 + x_4)$. While (42) does not model the individual effect of either x_3 or x_4 on y as well as (37)–(38), it does include dependency between x_3 and x_4. Furthermore, x_1 and x_2 are modeled equivalently (i.e., less tightly than individual constraints). Analyzing partitions with unequal size reveals similar tradeoffs. This section shows that the proposed formulation selects a subset of constraints from the convex hull formulation (26), revealing a strong tradeoff between modeling the effect of individual variables well vs the effect of many variables weakly.

Remark. *While the above 4-D case suggests "unbalanced" partitions may be favorable, it is difficult to develop a small example with $N << \eta$. Our computational results confirm that "balanced" partitions perform better.*

D. Strategies for Selecting Partitions

The above rationale strongly motivates selecting partitions that result in a model that treats input variables (approximately) equivalently for the practical case of $N << \eta$. Specifically, we seek to evenly distribute tradeoffs in model tightness among input variables. Section III-C suggests a reasonable approach is to select partitions such that the weights in each are approximately the same (weights are fixed during optimization). We propose two such strategies below.

Partitions of Equal Size. One strategy we propose to achieve this is to create partitions of *equal size*, i.e., $|\mathbb{S}_1| = |\mathbb{S}_2| = ... = |\mathbb{S}_N|$ (note that they may differ by up to one if η is not a multiple of N). The indices are then assigned to partitions to keep the weights in each partition as close as possible. This is accomplished by first sorting the weights $\boldsymbol{w}$, then distributing them evenly among the partitions (`array_split(argsort(`$\boldsymbol{w}$`), `N`)` in Numpy).

Partitions of Equal Range. A second strategy is to partition with *equal range*, i.e., $\underset{i\in\mathbb{S}_1}{\text{range}}(w_i) = ... = \underset{i\in\mathbb{S}_N}{\text{range}}(w_i)$. We define thresholds $\boldsymbol{v} \in \mathbb{R}^{N+1}$ such that v_1 and v_{N+1} are $\min(\boldsymbol{w})$ and $\max(\boldsymbol{w})$. To reduce the effect of outliers, we define v_2 and v_N as the 0.05 and 0.95 quantiles of $\boldsymbol{w}$, respectively. The remaining elements of $\boldsymbol{v}$ are distributed evenly in (v_2, v_N). Indices $i \in \{1, ..., \eta\}$ are then assigned to the subset $\mathbb{S}_n : w_i \in [v_n, v_{n+1})$. This strategy requires $N \geq 3$, but, for a symmetrically distributed weight vector, $\boldsymbol{w}$, two partitions of equal size are also of equal range.

We compare our proposed strategies against the following:

Random Partitions. Partitions are created by assigning indices $\{1, ..., \eta\}$ randomly to partitions $\mathbb{S}_1, ..., \mathbb{S}_N$.

Uneven Magnitudes. Weights are sorted by decreasing magnitude and "dealt" to partitions in a snake-draft order.

IV. Experiments

All computational experiments were performed on a 3.2 GHz Intel Core i7-8700 CPU (12 threads) with 16 GB memory. Models were implemented and solved using Gurobi v 9.1 [25]. The LP algorithm was set to dual simplex, `cuts` $= 1$ (moderate cut generation), `TimeLimit` $= 3600s$, and default termination criteria were used. We set parameter `MIPFocus` $= 3$ to ensure consistent solution approaches across formulations.

Neural Networks. We trained several NNs on MNIST [26] and CIFAR-10 [27], including both fully connected NNs and convolutional NNs (CNNs). Dense models are denoted by $n_{\text{Layers}} \times n_{\text{Hidden}}$ and comprise $n_{\text{Layers}} \times n_{\text{Hidden}}$ hidden plus 10 output nodes. CNN2 is based on 'ConvSmall' of the ERAN dataset [28]: {`Conv2D`(16, (4,4), (2,2)), `Conv2D`(32, (4,4), (2,2)), `Dense`(100), `Dense`(10)}. CNN1 halves the channels in each convolutional layer: {`Conv2D`(8, (4,4), (2,2)), `Conv2D`(16, (4,4), (2,2)), `Dense`(100), `Dense`(10)}. The implementations of CNN1/CNN2 have 1,852/3,604 and 2,476/4,852 nodes for MNIST and CIFAR-10, respectively. NNs are implemented in PyTorch [29] and obtained using standard training (i.e., without regularization or methods to improve robustness).

A. Optimal Adversary Results

The ***optimal adversary*** problem takes a target image $\bar{\boldsymbol{x}}$, its correct label j, and an adversarial label k, and finds the image within a range of perturbations maximizing the difference in predictions. Mathematically, this is formulated as $\max_{\boldsymbol{x}}(f_k(\boldsymbol{x}) - f_j(\boldsymbol{x})); x \in \mathcal{X}$, where f_k and f_j correspond to the k- and j-th elements of the NN output layer, respectively, and $\mathcal{X}$ defines the domain of perturbations. We consider perturbations defined by the ℓ_1-norm ($||\boldsymbol{x} - \bar{\boldsymbol{x}}||_1 \leq \epsilon_1 \in \mathbb{R}$), which promotes sparse perturbations [30]. For each dataset, we use the first 100 images from the corresponding test dataset and randomly selected adversarial labels (the same 100 verification problems are used for models trained on the same dataset).

Table I gives the optimal adversary results. Perturbations ϵ_1 were selected such that some big-M problems were solvable within 3600s (problems become more difficult as ϵ_1 increases). While formulations with both two and four partitions consistently outperform big-M in terms of problems solved and solution times, the best choice of N is problem-dependent. For instance, the 4-partition formulation performs best for the 2×100 network trained on CIFAR-10; Figure 2 shows the number of problems solved for this case. The 2-partition formulation is best for easy problems, but is soon overtaken by larger values of N. Intuitively, simpler problems are solved with fewer nodes in a branch-and-bound tree and benefit more from smaller subproblems. Performance declines again near $N \geq 7$, supporting observations that the convex-hull formulation ($N = \eta$) is not always best [13]. All partition-based formulations consistently outperform big-M ($N = 1$).

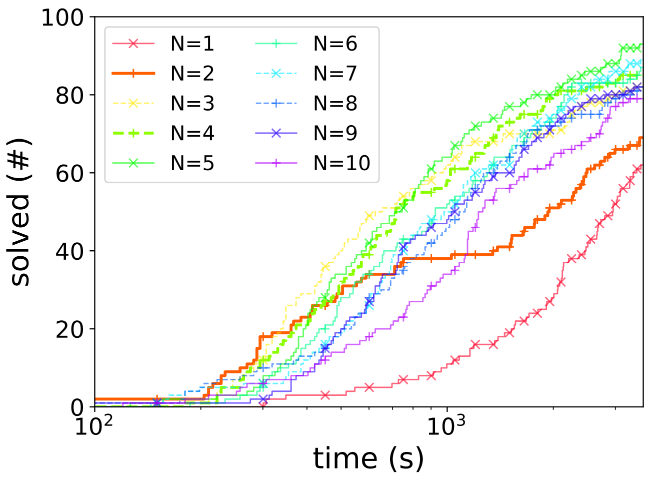


Fig. 2. Number solved vs run time for optimal adversary problems ($\epsilon = 5$) for the CIFAR-10, 2×100 model for various values of N. Each line shows 100 runs. $N = 1$ (equivalent to big-M) performs the worst; $N = 2$ performs well for easier problems; and intermediate values of N balance tradeoffs between model size and tightness well. Performance begins to decline near $N \geq 7$.

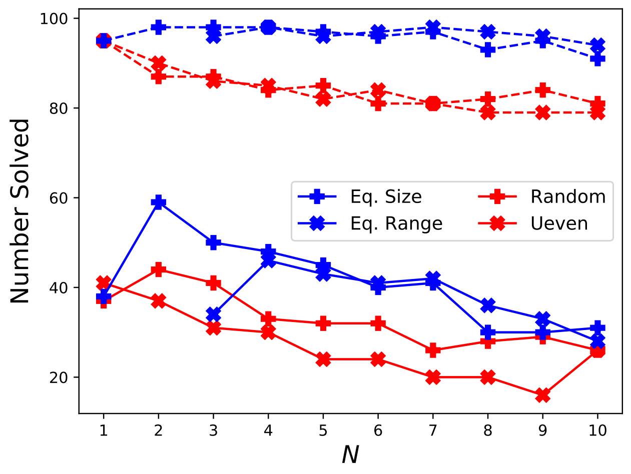
Partitioning Strategies. Figure 3 shows the result of the input partitioning strategies from Section III-D on the MNIST 2×100 model for varying N. Both proposed strategies (blue) outperform formulations with random and uneven partitions (red). With OBBT, big-M ($N = 1$) outperforms partition-based formulations when partitions are selected poorly. Figure 3 also shows the same tradeoff as Figure 2: our formulations perform best for some intermediate N, while the random/uneven partitions worsen with increasing N. The equal range strategy performs best for large N.

Optimization-Based Bounds Tightening. OBBT was implemented by tightening bounds for all z_n^b. We found

TABLE I
NUMBER OF SOLVED (IN 3600S) OPTIMAL ADVERSARY PROBLEMS AND AVERAGE SOLVE TIMES FOR BIG-M VS $N = \{2, 4\}$ EQUAL-SIZE PARTITIONS. AVERAGE TIMES COMPUTED FOR PROBLEMS SOLVED BY ALL 3 FORMULATIONS. GREY INDICATES FORMULATIONS STRICTLY OUTPERFORMING BIG-M.

Dataset	Model	ϵ_1	Big-M		2 Partitions		4 Partitions	
			solved(#)	avg.time(s)	solved(#)	avg.time(s)	solved(#)	avg.time(s)
MNIST	2×50	5	100	57.6	100	**42.7**	100	83.9
	2×50	10	97	431.8	98	**270.5**	98	398.4
	2×100	2.5	92	525.2	100	**285.1**	94	553.9
	2×100	5	32	1473.7	59	**494.6**	48	988.9
	CNN1*	0.25	68	1099.7	86	**618.8**	87	840.0
	CNN1*	0.5	2	2293.2	16	**1076.0**	11	2161.2
CIFAR-10	2×100	5	62	1982.3	69	1083.4	85	**752.8**
	2×100	10	23	2319.0	28	1320.2	34	**1318.1**

*OBBT performed on all NN nodes



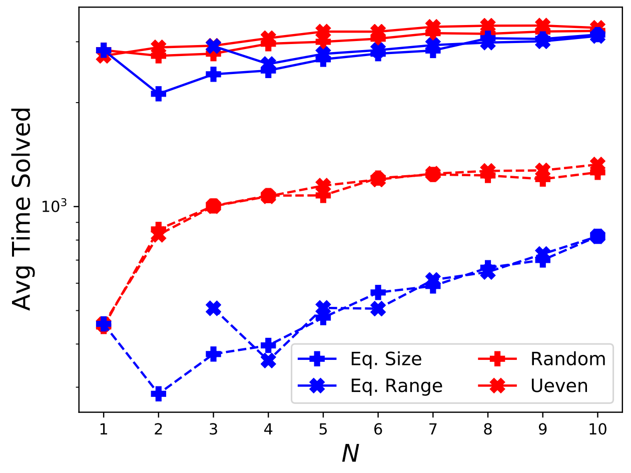


Fig. 3. Number solved (top) and solution times (bottom) for optimal adversary problems for MNIST 2×100 ($\epsilon = 5$). Each point shows 100 runs, max time of 3600s. Dashed lines show runs with OBBT. The equal range strategy requires $N \geq 3$. Our proposed (blue) partitioning strategies solve more problems (top) faster (bottom) than random and uneven partitions (red)

that adding the bounds from the 1-partition model (i.e., bounds on $\sum z_n^b$) improved all models, as they account for dependencies among all inputs. Therefore these bounds were used in all models, resulting in $2N+2$ LPs per node ($N \geq 2$). We limited the OBBT LPs to 5s; interval bounds were used if an LP was not solved. Figure 3 shows that OBBT greatly improves the optimization performance of all formulations. OBBT problems for each layer are independent and could, in practice, be solved in parallel. Therefore, at a minimum, OBBT requires the sum of max solution times found in each layer (5s $\times$ # layers in this case). This represents an avenue to significantly improve MILP optimization of NNs via parallelization. In contrast, parallelizing branch-and-bound is known to be challenging and gives limited benefits [31], [32].

B. Verification Results

The ***verification*** problem is defined similarly to the optimal adversary problem, but terminates when the sign of the objective function is known (i.e., the lower/upper bounds of the MILP have the same sign). This problem is typically solved for perturbations defined by the ℓ_∞-norm ($||\boldsymbol{x} - \bar{\boldsymbol{x}}||_\infty \leq \epsilon_\infty \in \mathbb{R}$). Here, problems are difficult for moderate ϵ_∞: at large ϵ_∞ a mis-classified example (positive objective) is easily found. Several verfication tools rely on an underlying big-M formulation, e.g., MIPVerify [4], NSVerify [33], making big-M an especially relevant point of comparison.

Owing to the early termination, larger NNs can be tested compared to the optimal adversary problems. We turned off cuts (`cuts`$= 0$) for the partition-based formulations, as the models are relatively tight over the box-domain perturbations and do not seem to benefit from additional cuts. On the other hand, removing cuts improved some problems using big-M and worsened others. Results for the verification problem are presented in Table II. The partition-based formulations again generally outperform big-M ($N = 1$), except for a few of the 4-partition problems.

C. Minimally Distorted Adversary Results

In a similar vein as [34], we define the ***ℓ_1-minimally distorted adversary*** problem: given a target image $\bar{\boldsymbol{x}}$ and its correct label j, find the smallest perturbation over which the NN predicts an adversarial label k. We formulate this as $\min_{\epsilon_1, \boldsymbol{x}} \epsilon_1; ||\boldsymbol{x} - \bar{\boldsymbol{x}}||_1 \leq \epsilon_1; f_k(\boldsymbol{x}) \geq f_j(\boldsymbol{x})$. The adversarial label k is selected as the second-likeliest class of the target image. Figure 4 illustrates that the ℓ_1-norm promotes sparse perturbations, unlike the ℓ_∞-norm.

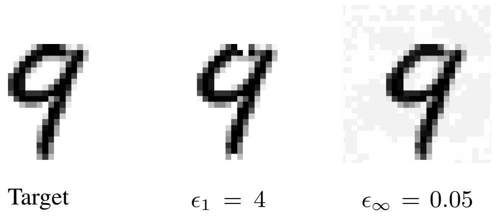


Fig. 4. Sample ℓ_1- vs ℓ_∞-based minimally distorted adversaries for the MNIST 2×50 model. The adversarial label (k) is '4.'

Table III presents results for the minimally distorted adversary problems. As input domains are unbounded, these problems are considerably more difficult than the above *optimal adversary* problems. Therefore, only smaller MNIST networks were manageable (with OBBT) for all formulations. Again, all partition-based formulations

TABLE II

NUMBER OF SOLVED (IN 3600S) VERIFICATION PROBLEMS AND AVERAGE SOLVE TIMES FOR BIG-M VS $N = \{2, 4\}$ EQUAL-SIZE PARTITIONS. AVERAGE TIMES COMPUTED FOR PROBLEMS SOLVED BY ALL 3 FORMULATIONS. OBBT WAS PERFORMED FOR ALL PROBLEMS. GREY INDICATES FORMULATIONS STRICTLY OUTPERFORMING BIG-M.

Dataset	Model	ϵ_∞	Big-M		2 Partitions		4 Partitions	
			solved(#)	avg.time(s)	solved(#)	avg.time(s)	solved(#)	avg.time(s)
MNIST	CNN1	0.050	82	198.5	92	**27.3**	90	52.4
	CNN1	0.075	30	632.5	52	**139.6**	42	281.6
	CNN2	0.075	21	667.1	36	**160.7**	31	306.0
	CNN2	0.100	1	505.3	5	**134.7**	5	246.3
CIFAR-10	CNN1	0.007	99	100.6	100	**25.9**	99	45.4
	CNN1	0.010	98	85.1	100	**21.1**	100	37.5
	CNN2	0.007	80	1016.7	95	**412.8**	68	1712.7
	CNN2	0.010	40	2246.7	72	**859.9**	35	2449.4

TABLE III

NUMBER OF SOLVED (IN 3600S) ℓ_1-MINIMALLY DISTORTED ADVERSARY PROBLEMS AND AVERAGE SOLVE TIMES FOR BIG-M VS $N = \{2, 4\}$ EQUAL-SIZE PARTITIONS. AVERAGE TIMES AND ϵ_1 ARE COMPUTED FOR PROBLEMS SOLVED BY ALL 3 FORMULATIONS. OBBT WAS PERFORMED FOR ALL PROBLEMS. GREY INDICATES PARTITION FORMULATIONS STRICTLY OUTPERFORMING BIG-M.

Dataset	Model	avg(ϵ_1)	Big-M		2 Partitions		4 Partitions	
			solved(#)	avg.time(s)	solved(#)	avg.time(s)	solved(#)	avg.time(s)
MNIST	2×50	6.51	52	675.0	93	**150.9**	89	166.6
	2×75	4.41	16	547.3	37	**310.5**	31	424.0
	2×100	2.73	7	710.8	13	**572.9**	10	777.9

consistently outperform big-M, solving more problems and in less time.

V. CONCLUSIONS

This work presented MILP formulations for ReLU NNs that can balance having both a tight relaxation and manageable size. The approach strategically partitions node inputs and forms the convex hull over said partitions: we presented theoretical and computational motivations for obtaining good partitions for ReLU nodes. Furthermore, our approach expands the benefits of OBBT, which, unlike conventional MILP tools, can easily be parallelized. Results on three classes of optimization tasks show that the proposed formulations consistently outperform standard MILP encodings, allowing us to solve 25% more of the considered problems. A >2X speedup is achieved on average for solved problems.

ACKNOWLEDGMENTS

This work was supported by Engineering & Physical Sciences Research Council (EPSRC) Fellowships to CT and RM (grants EP/T001577/1 and EP/P016871/1), an Imperial College Research Fellowship to CT, a Royal Society Newton International Fellowship (NIF\R1\182194) to JK, a grant by the Swedish Cultural Foundation in Finland to JK, and a PhD studentship funded by BASF to AT.

REFERENCES

[1] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NASA Formal Methods Symposium*. Springer, 2018, pp. 121–138.

[2] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward ReLU neural networks," *arXiv preprint arXiv:1706.07351*, 2017.

[3] G. Wu, B. Say, and S. Sanner, "Scalable planning with deep neural network learned transition models," *Journal of Artificial Intelligence Research*, vol. 68, pp. 571–606, 2020.

[4] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *arXiv preprint arXiv:1711.07356*, 2017.

[5] B. Grimstad and H. Andersson, "ReLU networks as surrogate models in mixed-integer linear programs," *Computers & Chemical Engineering*, vol. 131, p. 106580, 2019.

[6] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, "Efficient verification of ReLU-based neural networks via dependency analysis," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3291–3299.

[7] R. Bunel, I. Turkaslan, P. H. Torr, P. Kohli, and M. P. Kumar, "A unified view of piecewise linear neural network verification," *arXiv preprint arXiv:1711.00455*, 2017.

[8] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2017, pp. 251–268.

[9] T. Serra, A. Kumar, and S. Ramalingam, "Lossless compression of deep neural networks," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2020, pp. 417–430.

[10] T. Serra, C. Tjandraatmadja, and S. Ramalingam, "Bounding and counting linear regions of deep neural networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4558–4566.

[11] M. Fischetti and J. Jo, "Deep neural networks and mixed integer linear optimization," *Constraints*, vol. 23, no. 3, pp. 296–309, 2018.

[12] M. Conforti, G. Cornuéjols, and G. Zambelli, "Integer programming, volume 271 of Graduate Texts in Mathematics," 2014.

[13] R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, and J. P. Vielma, "Strong mixed-integer programming formulations for trained neural networks," *Mathematical Programming*, pp. 1–37, 2020.

[14] J. Kronqvist, R. Misener, and C. Tsay, "Between steps: Intermediate relaxations between big-M and convex hull formulations," *arXiv preprint arXiv:2101.12708*, 2021.

[15] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5286–5295.

[16] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2017, pp. 269–286.

[17] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon, "Towards fast computation of certified robustness for ReLU networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5276–5285.

[18] A. Raghunathan, J. Steinhardt, and P. S. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," in *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc., 2018, pp. 10 877–10 887.

[19] S. Dathathri, K. Dvijotham, A. Kurakin, A. Raghunathan, J. Uesato, R. Bunel, S. Shankar, J. Steinhardt, I. Goodfellow, P. Liang *et al.*, "Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming," *arXiv preprint arXiv:2010.11645*, 2020.

[20] R. Bunel, A. De Palma, A. Desmaison, K. Dvijotham, P. Kohli, P. Torr, and M. P. Kumar, "Lagrangian decomposition for neural network verification," in *Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 370–379.

[21] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "Boosting robustness certification of neural networks." in *International Conference on Learning Representations*, 2019.

[22] G. Singh, R. Ganvir, M. Püschel, and M. Vechev, "Beyond the single neuron convex barrier for neural network certification," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 098–15 109.

[23] E. Balas, *Disjunctive Programming*. Springer International Publishing, 2018.

[24] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks." in *Conference on Uncertainty in Artificial Intelligence*, vol. 1, no. 2, 2018, p. 3.

[25] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[26] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[27] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[28] G. Singh, J. Maurer, C. Müller, M. Mirman, T. Gehr, A. Hoffmann, P. Tsankov, D. Drachsler Cohen, M. Püschel, and M. Vechev, "ERAN verification dataset." [Online]. Available: https://github.com/eth-sri/eran

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[30] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, "Ead: elastic-net attacks to deep neural networks via adversarial examples," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[31] T. Achterberg and R. Wunderling, "Mixed integer programming: Analyzing 12 years of progress," in *Facets of Combinatorial Optimization*. Springer, 2013, pp. 449–481.

[32] T. Ralphs, Y. Shinano, T. Berthold, and T. Koch, "Parallel solvers for mixed integer linear optimization," in *Handbook of Parallel Constraint Reasoning*. Springer, 2018, pp. 283–336.

[33] M. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano, "Reachability analysis for neural agent-environment systems." in *International Conference on Principles of Knowledge Representation and Reasoning*, 2018, pp. 184–193.

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