Strong Mixed-Integer Programming Formulations for Trained Neural Networks

Ross Anderson[1], Joey Huchette[1](✉), Christian Tjandraatmadja[1], and Juan Pablo Vielma[2]

[1] Google Research, Cambridge, USA
{rander,jhuchette,ctjandra}@google.com
[2] MIT, Cambridge, USA
jvielma@mit.edu

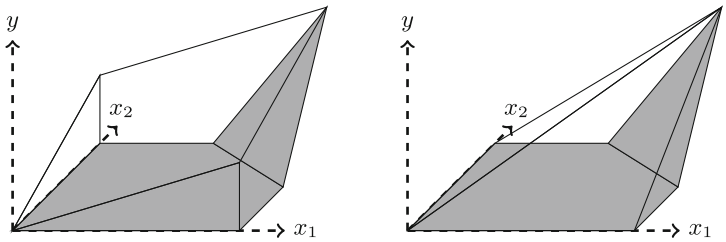

Fig. 1. The convex relaxation for a ReLU neuron using: **(Left)** existing MIP formulations, and **(Right)** the formulations presented in this paper.

Abstract. We present an ideal mixed-integer programming (MIP) formulation for a rectified linear unit (ReLU) appearing in a trained neural network. Our formulation requires a single binary variable and no additional continuous variables beyond the input and output variables of the ReLU. We contrast it with an ideal "extended" formulation with a linear number of additional continuous variables, derived through standard techniques. An apparent drawback of our formulation is that it requires an exponential number of inequality constraints, but we provide a routine to separate the inequalities in linear time. We also prove that these exponentially-many constraints are facet-defining under mild conditions. Finally, we study network verification problems and observe that dynamically separating from the exponential inequalities (1) is much more computationally efficient and scalable than the extended formulation, (2) decreases the solve time of a state-of-the-art MIP solver by a factor of 7 on smaller instances, and (3) nearly matches the dual bounds of a state-of-the-art MIP solver on harder instances, after just a few rounds of separation and in orders of magnitude less time.

Keywords: Mixed-integer programming · Formulations · Deep learning

© Springer Nature Switzerland AG 2019
A. Lodi and V. Nagarajan (Eds.): IPCO 2019, LNCS 11480, pp. 27–42, 2019.
https://doi.org/10.1007/978-3-030-17953-3_3

1 Introduction

Deep learning has proven immensely powerful at solving a number of important predictive tasks arising in areas such as image classification, speech recognition, machine translation, and robotics and control [27,35]. The workhorse model in deep learning is the feedforward network $\texttt{NN} : \mathbb{R}^{m_0} \to \mathbb{R}^{m_s}$ with rectified linear unit (ReLU) activation functions, for which $\texttt{NN}(x^0) = x^s$ is defined through

$$x^i_j = \texttt{ReLU}(w^{i,j} \cdot x^{i-1} + b^{i,j}) \tag{1}$$

for each layer $i \in [\![s]\!] \stackrel{\text{def}}{=} \{1, \ldots, s\}$ and $j \in [\![m_i]\!]$. Note that the input $x^0 \in \mathbb{R}^{m_0}$ might be high-dimensional, and that the output $x^s \in \mathbb{R}^{m_s}$ may be multivariate. In this recursive description, $\texttt{ReLU}(v) \stackrel{\text{def}}{=} \max\{0, v\}$ is the ReLU activation function, and $w^{i,j}$ and $b^{i,j}$ are the weights and bias of an affine function which are learned during the training procedure. Each equation in (1) corresponds to a single *neuron* in the network. Networks with any specialized linear transformations such as convolutional layers can be reduced to this model after training, without loss of generality.

There are numerous contexts in which one may want to solve an optimization problem containing a trained neural network such as NN. For example, such problems arise in deep reinforcement learning problems with high dimensional action spaces and where any of the cost-to-go function, immediate cost, or the state transition functions are learned by a neural network [3,19,40,44,55]. Alternatively, there has been significant recent interest in verifying the robustness of trained neural networks deployed in systems like self-driving cars that are incredibly sensitive to unexpected behavior from the machine learning model [15,43,48]. Relatedly, a string of recent work has used optimization over neural networks trained for visual perception tasks to *generate* new images which are "most representative" for a given class [42], are "dreamlike" [41], or adhere to a particular artistic style via neural style transfer [26].

1.1 MIP Formulation Preliminaries

In this work, we study mixed-integer programming (MIP) approaches for optimization problems containing trained neural networks. In contrast to heuristic or local search methods often deployed for the applications mentioned above, MIP offers a framework for producing provably optimal solutions. This is of particular interest in the verification problem, where rigorous dual bounds can guarantee robustness in a way that purely primal methods cannot.

We focus on constructing MIP formulations for the *graph* of ReLU neurons:

$$\text{gr}(\texttt{ReLU} \circ f; [L, U]) \stackrel{\text{def}}{=} \{\, (x, (\texttt{ReLU} \circ f)(x)) \mid L \leqslant x \leqslant U \,\}, \tag{2}$$

where $\circ$ is the standard function composition operator $(g \circ f)(x) = g(f(x))$. This substructure consists of a single ReLU activation function, taking as input an affine function $f(x) = w \cdot x + b$ over a η-dimensional box-constrained input

domain. The nonlinearity is handled by introducing an auxiliary binary variable z to indicate whether $(\texttt{ReLU} \circ f)(x) = 0$ or $(\texttt{ReLU} \circ f)(x) = f(x)$ for a given value of x. We focus on these particular substructures because we can readily produce a MIP formulation for the entire network as the composition of formulations for each individual neuron.[1]

A MIP formulation is *ideal* if the extreme points of its linear programming (LP) relaxation are integral. Ideal formulations are highly desirable from a computational perspective, and offer the strongest possible convex relaxation for the set being formulated [50].

Our main contribution is an ideal formulation for a single ReLU neuron with no auxiliary continuous variables and an exponential number of inequality constraints. We show that each of these exponentially-many constraints is facet-defining under very mild conditions. We also provide a simple linear-time separation routine to generate the most violated inequality from the exponential family. This formulation is derived by constructing an ideal extended formulation that uses η auxiliary continuous variables and projecting them out. We evaluate our methods computationally on verification problems for image classification networks trained on the MNIST digit dataset, where we observe that separating over these exponentially-many inequalities solves smaller instances faster than using Gurobi's default cut generation by a factor of 7, and (nearly) matches the dual bounds on larger instances in orders of magnitude less time.

1.2 Relevant Prior Work

In recent years a number of authors have used MIP formulations to model trained neural networks [14,16,20,25,32,38,44,46,47,49,55,56], mostly applying big-M formulation techniques to ReLU-based networks. When applied to a single neuron of the form (2), these big-M formulations will not be ideal or offer an exact convex relaxation; see Example 1 for an illustration. Additionally, a stream of literature in the deep learning community has studied convex relaxations in the original space of input/output variables x and y (or a dual representation thereof), primarily for verification tasks [9,22,23]. It has been shown that these convex relaxations are equivalent to those provided by the standard big-M MIP formulation, after projecting out the auxiliary binary variables (e.g. [46]). Moreover, some authors have investigated how to use convex relaxations within the training procedure in the hopes of producing neural networks with a priori robustness guarantees [21,53,54].

Beyond MIP and convex relaxations, a number of authors have investigated other algorithmic techniques for modeling trained neural networks in optimization problems, drawing primarily from the satisfiability, constraint programming, and global optimization communities [7,8,33,37,45]. Another intriguing direction studies restrictions to the space of models that may make the optimization problem over the network inputs simpler: for example, the classes of binarized [34] or input convex [1] neural networks.

[1] Further analysis of the interactions between neurons can be found in the full-length version of this extended abstract [2].

Broadly, our work fits into a growing body of research in prescriptive analytics and specifically the "predict, then optimize" framework, which considers how to embed trained machine learning models into optimization problems [11,12,17,18,24,28,39]. Additionally, the formulations presented below have connections with existing structures studied in the MIP and constraint programming community like indicator variables and on/off constraints [4,10,13,29,30].

1.3 Starting Assumptions and Notation

We will assume that $-\infty < L_i < U_i < \infty$ for each input component i. While a bounded input domain will make the formulations and analysis considerably more difficult than the unbounded setting (see [4] for a similar phenomenon), it ensures that standard MIP representability conditions are satisfied (e.g. [50, Sect. 11]). Furthermore, variable bounds are natural for many applications (for example in verification problems), and are absolutely essential for ensuring reasonable dual bounds.

Define $\breve{L}, \breve{U} \in \mathbb{R}^\eta$ such that, for each $i \in [\![\eta]\!]$,

$$\breve{L}_i = \begin{cases} L_i & \text{if } w_i \geqslant 0 \\ U_i & \text{if } w_i < 0 \end{cases} \quad \text{and} \quad \breve{U}_i = \begin{cases} U_i & \text{if } w_i \geqslant 0 \\ L_i & \text{if } w_i < 0 \end{cases}.$$

This definition implies that $w_i \breve{L}_i \leqslant w_i \breve{U}_i$ for each i, which simplifies the handling of negative weights $w_i < 0$. Take the values $M^+(f) \stackrel{\text{def}}{=} \max_{\tilde{x} \in [L,U]} f(\tilde{x}) \equiv w \cdot \breve{U} + b$ and $M^-(f) \stackrel{\text{def}}{=} \min_{\tilde{x} \in [L,U]} f(\tilde{x}) \equiv w \cdot \breve{L} + b$. Define $\operatorname{supp}(w) \stackrel{\text{def}}{=} \{\, i \in [\![\eta]\!] \mid w_i \neq 0 \,\}$. Finally, take $\mathbb{R}_{\geqslant 0} \stackrel{\text{def}}{=} \{\, x \in \mathbb{R} \mid x \geqslant 0 \,\}$ as the nonnegative orthant.

We say that *strict activity* holds for a given ReLU neuron $\operatorname{gr}(\texttt{ReLU} \circ f; [L,U])$ if $M^-(f) < 0 < M^+(f)$, or in other words, if $\operatorname{gr}(\texttt{ReLU} \circ f; [L,U])$ is not equal to either $\operatorname{gr}(0; [L,U])$ or $\operatorname{gr}(f; [L,U])$. We assume for the remainder that strict activity holds for each ReLU neuron. This assumption is not onerous, as otherwise, the nonlinearity can be replaced by an affine function (either 0 or $w \cdot x + b$). Moreover, strict activity can be verified or disproven in time linear in η.

2 The ReLU Neuron

The ReLU is the workhorse of deep learning models: it is easy to reason about, introduces little computational overhead, and despite its simple structure is nonetheless capable of articulating complex nonlinear relationships.

2.1 A Big-M Formulation

A standard big-M formulation for $\operatorname{gr}(\texttt{ReLU} \circ f; [L,U])$ is:

$$y \geqslant f(x) \tag{3a}$$

$$y \leqslant f(x) - M^-(f) \cdot (1 - z) \tag{3b}$$

$$y \leqslant M^+(f) \cdot z \tag{3c}$$

$$(x, y, z) \in [L,U] \times \mathbb{R}_{\geqslant 0} \times \{0,1\}. \tag{3d}$$

This is the formulation used recently in the bevy of papers referenced in Sect. 1.2. Unfortunately, this formulation is not necessarily ideal, as illustrated by the following example.

Example 1. If $f(x) = x_1 + x_2 - 1.5$, formulation (3a–3d) for $\mathrm{gr}(\texttt{ReLU} \circ f; [0,1]^2)$ is

$$y \geqslant x_1 + x_2 - 1.5 \tag{4a}$$
$$y \leqslant x_1 + x_2 - 1.5 + 1.5(1 - z) \tag{4b}$$
$$y \leqslant 0.5z \tag{4c}$$
$$(x, y, z) \in [0,1]^2 \times \mathbb{R}_{\geqslant 0} \times [0,1] \tag{4d}$$
$$z \in \{0, 1\}. \tag{4e}$$

The point $(\hat{x}, \hat{y}, \hat{z}) = ((1,0), 0.25, 0.5)$ is feasible for the LP relaxation (4a–4d); however, $(\hat{x}, \hat{y}) \equiv ((1,0), 0.25)$ is not in $\mathrm{Conv}(\mathrm{gr}(\texttt{ReLU} \circ f; [0,1]^2))$, and so the formulation does not offer an exact convex relaxation (and, hence, is not ideal). See Fig. 1 for an illustration: on the left, of the big-M formulation projected to (x, y)-space, and on the right, the tightest possible convex relaxation.

The integrality gap of (3a–3d) can be arbitrarily bad, even in fixed dimension η.

Example 2. Fix $\gamma \in \mathbb{R}_{\geqslant 0}$ and even $\eta \in \mathbb{N}$. Take the affine function $f(x) = \sum_{i=1}^{\eta} x_i$, the input domain $[L, U] = [-\gamma, \gamma]^\eta$, and the point $\hat{x} = \gamma \cdot (1, -1, \cdots, 1, -1)$ as a scaled vector of alternating ones and negative ones. We can check that $(\hat{x}, \hat{y}, \hat{z}) = (\hat{x}, \frac{1}{2}\gamma\eta, \frac{1}{2})$ is feasible for the LP relaxation of the big-M formulation (3a–3d). Additionally, $f(\hat{x}) = 0$, and for any $\tilde{y}$ such that $(\hat{x}, \tilde{y}) \in \mathrm{Conv}(\mathrm{gr}(\texttt{ReLU} \circ f; [L, U]))$, then $\tilde{y} = 0$ necessarily. Therefore, there exists a fixed point $\hat{x}$ in the input domain where the tightest possible convex relaxation (for example, from an ideal formulation) is exact, but the big-M formulation deviates from this value by at least $\frac{1}{2}\gamma\eta$.

Intuitively, this example suggests that the big-M formulation is particularly weak around the boundary of the input domain, as it cares only about the value $f(x)$ of the affine function, and not the particular input value x.

2.2 An Ideal Extended Formulation

It is possible to produce an ideal *extended* formulation for the ReLU neuron by introducing auxiliary continuous variables. The "multiple choice" formulation is

$$(x, y) = (x^0, y^0) + (x^1, y^1) \tag{5a}$$
$$y^0 = 0 \geqslant w \cdot x^0 + b(1 - z) \tag{5b}$$
$$y^1 = w \cdot x^1 + bz \geqslant 0 \tag{5c}$$
$$L(1 - z) \leqslant x^0 \leqslant U(1 - z) \tag{5d}$$
$$Lz \leqslant x^1 \leqslant Uz \tag{5e}$$
$$z \in \{0, 1\}, \tag{5f}$$

is an ideal extended formulation for piecewise linear functions [52]. It can alternatively be derived from techniques introduced by Balas [5,6]. Although the multiple choice formulation offers the tightest possible convex relaxation for a single neuron, it requires a copy x^0 of the input variables (note that it is straightforward to use Eq. (5a) to eliminate the second copy x^1). This means that when the multiple choice formulation is applied to every neuron in the network to formulate NN, the total number of continuous variables required is $m_0 + \sum_{i=1}^{r}(m_{i-1}+1)m_i$ (using the notation of (1), where m_i is the number of neurons in layer i). In contrast, the big-M formulation requires only $m_0 + \sum_{i=1}^{r} m_i$ continuous variables to formulate the entire network. As we will see in Sect. 3.2, the quadratic growth in size of the extended formulation can quickly become burdensome. Additionally, a folklore observation in the MIP community is that multiple choice formulations tend to not perform as well as expected in simplex-based branch-and-bound algorithms, likely due to degeneracy introduced by the block structure [51].

2.3 An Ideal Non-extended Formulation

We now present a non-extended ideal formulation for the ReLU neuron, stated only in terms of the original variables (x, y) and the single binary variable z. Put another way, it is the strongest possible tightening that can be applied to the big-M formulation (3a–3d) and so matches the strength of the multiple choice formulation without the additional continuous variables.

Proposition 1. *Take some affine function $f(x) = w \cdot x + b$ over input domain $[L, U]$. The following is an ideal formulation for* $\mathrm{gr}(\texttt{ReLU} \circ f; [L, U])$*:*

$$y \geqslant w \cdot x + b \tag{6a}$$
$$y \leqslant \sum_{i \in I} w_i(x_i - \breve{L}_i(1 - z)) + \left(b + \sum_{i \notin I} w_i \breve{U}_i\right) z \quad \forall I \subseteq \mathrm{supp}(w) \tag{6b}$$
$$(x, y, z) \in [L, U] \times \mathbb{R}_{\geqslant 0} \times \{0, 1\} \tag{6c}$$

Proof. See Appendix A.1. □

Furthermore, each of the exponentially-many inequalities in (6b) is necessary.

Proposition 2. *Each inequality in* (6b) *is facet-defining.*

Proof. See Appendix A.2. □

We require the assumption of strict activity above, as introduced in Sect. 1.3. Under the same condition, it is also possible to show that (6a) is facet-defining, but we omit it in this extended abstract for brevity. As a result of this and Proposition 2, the formulation (6a–6c) is minimal (modulo variable bounds).

The proof of Proposition 2 offers a geometric interpretation of the facets induced by (6b). Each facet is a convex combination of two faces: an $(\eta - |I|)$-dimensional face consisting of all feasible points with $z = 0$ and $x_i = \breve{L}_i$ for all $i \in [\![\eta]\!] \setminus I$, and an $|I|$-dimensional face consisting of all feasible points with $z = 1$ and $x_i = \breve{U}_i$ for all $i \in I$.

It is also possible to separate from the family (6b) in time linear in η.

Proposition 3. *Take a point* $(\hat{x}, \hat{y}, \hat{z}) \in [L, U] \times \mathbb{R}_{\geqslant 0} \times [0, 1]$*, along with the set*

$$\hat{I} = \left\{ i \in \operatorname{supp}(w) \;\middle|\; w_i \hat{x}_i < w_i \left(\breve{L}_i (1 - \hat{z}) + \breve{U}_i \hat{z} \right) \right\}.$$

If any constraint in the family (6b) *is violated at* $(\hat{x}, \hat{y}, \hat{z})$*, then the one corresponding to* $\hat{I}$ *is the most violated.*

Proof. Follows from inspecting the family (6b): each has the same left-hand-side, and so to maximize violation, it suffices to select the subset I that minimizes the right-hand-side. This can be performed in a separable manner, independently for each component $i \in \operatorname{supp}(w)$, giving the result. □

Observe that the inequalities (3b) and (3c) are equivalent to those in (6b) with $I = \operatorname{supp}(w)$ and $I = \varnothing$, respectively (modulo components i with $w_i = 0$). This suggests an iterative scheme to produce strong relaxations for ReLU neurons: start with the big-M formulation (3a–3d), and use Proposition 3 to separate strengthening inequalities from the exponential family (6b) as they are needed. We evaluate this approach in the following computational study.

3 Computational Experiments

To conclude the work, we study the strength of the ideal formulations presented in Sect. 2 for individual ReLU neurons. We study the verification problem on image classification networks trained on the canonical MNIST digit dataset [36]. We train a neural network $f : [0, 1]^{28 \times 28} \to \mathbb{R}^{10}$, where the 10 outputs correspond to the logits for each of the digits from 0 to 9. Given a labeled image $\tilde{x} \in [0, 1]^{28 \times 28}$, our goal is to prove or disprove the existence of a perturbation of $\tilde{x}$ such that the neural network f produces a wildly different classification result. If $f(\tilde{x})_i = \max_{j=1}^{10} f(\tilde{x})_j$, then image $\tilde{x}$ is placed in class i. To evaluate robustness around $\tilde{x}$ with respect to class j, we can solve the following optimization problem for some small constant $\epsilon > 0$:

$$\max_{a : ||a||_\infty \leqslant \epsilon} f(\tilde{x} + a)_j - f(\tilde{x} + a)_i.$$

If the optimal solution (or a valid dual bound thereof) is less than zero, this verifies that our network is robust around $\tilde{x}$ in the sense that we cannot produce a small perturbation that will flip the classification from i to j.

We train a smaller and a larger model, each with two convolutional layers with ReLU activation functions, feeding into a dense layer of ReLU neurons, and then a final dense linear layer. TensorFlow pseudocode specifying the two network architectures is included in Fig. 2. We generate 100 instances for each network by randomly selecting images $\tilde{x}$ with true label i from the test data, along with a random target adversarial class $j \neq i$. Note that we make no attempts to utilize recent techniques that train the networks to be verifiable [21,53,54,56].

```
input = placeholder(float32, shape=(28,28))
conv1 = conv2d(input, filters=4, kernel_size=4,
               strides=(2,2), activation=relu, use_bias=True)
conv2 = conv2d(conv1, filters=4, kernel_size=4,
               strides=(2,2), use_bias=True)
flatten = reshape(conv2, [5*5*4])
dense = dense(flatten, 16, activation=relu, use_bias=True)
logits = dense(dense, 10, use_bias=True)
```

(a) Smaller ReLU network.

```
input = placeholder(float32, shape=(28,28))
conv1 = conv2d(input, filters=16, kernel_size=4,
               strides=(2,2), activation=relu, use_bias=True)
conv2 = conv2d(conv1, filters=32, kernel_size=4,
               strides=(2,2), activation=relu, use_bias=True)
flatten = reshape(conv2, [5*5*32])
dense = dense(flatten, 100, activation=relu, use_bias=True)
logits = dense(dense, 10, use_bias=True)
```

(b) Larger ReLU network.

Fig. 2. TensorFlow pseudocode specifying the two network architectures used.

For all experiments, we use the Gurobi v7.5.2 solver, running with a single thread on a machine with 128 GB of RAM and 32 CPUs at 2.30 GHz. We use a time limit of 30 min (1800 s) for each run. We perform our experiments using the `tf.opt` package for optimization over trained neural networks; `tf.opt` is under active development at Google, with the intention to open source the project in the future. Below, the *big-M +* (6b) method is the big-M formulation (3a–3d) paired with separation[2] over the exponential family (6b), and with Gurobi's cutting plane generation turned off. Similarly, the *big-M* and the *extended* methods are the big-M formulation (3a–3d) and the extended formulation (5a– 5f) respectively, with default Gurobi settings. Finally, the *big-M + no cuts* method turns off Gurobi's cutting plane generation without adding separation over (6b).

[2] We use cut callbacks in Gurobi to inject separated inequalities into the cut loop. While this offers little control over when the separation procedure is run, it allows us to take advantage of Gurobi's sophisticated cut management implementation.

3.1 Small ReLU Network

We start with a smaller ReLU network whose architecture is depicted in TensorFlow pseudocode in Fig. 2a. The model attains 97.2% test accuracy. We select a perturbation ball radius of $\epsilon = 0.1$. We report the results in Table 1 and in Fig. 3. The big-M + (6b) method solves 7 times faster on average than the big-M formulation. Indeed, for 79 out of 100 instances the big-M method does not prove optimality after 30 min, and it is never the fastest choice (the "win" column). Moreover, the big-M + no cuts times out on every instance, implying that using *some* cuts is important. The extended method is roughly 5 times slower than the big-M + (6b) method, but only exceeds the time limit on 19 instances, and so is substantially more reliable than the big-M method for a network of this size. From this, we conclude that the additional strength offered by the ideal formulations (5a–5f) and (6a–6c) can offer substantial computational improvement over the big-M formulation (3a–3d).

3.2 Larger ReLU Network

Now we turn to the larger ReLU network described in Fig. 2b. The trained model attains 98.5% test accuracy. We select a perturbation ball radius of $\epsilon = 10/256$.

Table 1. Results for smaller network. Shifted geometric mean for time and optimality gap taken over 100 instances (shift of 10 and 1, respectively). The "win" column is the number of (solved) instances on which the method is the fastest.

Method	Time (s)	Optimality gap	Win
big-M + (6b)	174.49	0.53%	81
big-M	1233.49	6.03%	0
big-M + no cuts	1800.00	125.6%	0
Extended	890.21	1.26%	6

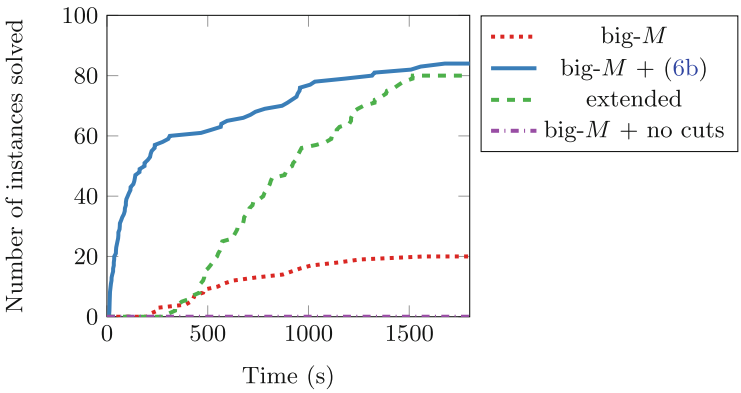

Fig. 3. Number of small network instances solved within a given amount of time. Curves to the upper left are better, with more instances solved in less time.

For these larger networks, we eschew solving the problems to optimality and focus on the quality of the dual bound available at the root node. As Gurobi does not reliably produce feasible primal solutions for these larger instances, we turn off primal heuristics and compare the approaches based on the "verification gap", which measures how far the dual bound is from proving robustness (i.e. an objective value of 0). To evaluate the quality of a dual bound, we measure the "improvement percentage" $\frac{\texttt{big_M_bound} - \texttt{other_bound}}{\texttt{big_M_bound}}$, where our baseline for comparison, `big_M_bound`, is the bound from the big-M + no cuts method, and `other_bound` is the dual bound being compared.

Table 2. Results at the root node for larger network. Shifted geometric mean of bound, time, and improvement over 100 instances (shift of 10).

Method	Bound	Time (s)	Improvement
big-M + no cuts	302.03	3.08	–
big-M + (6b)	254.95	8.13	15.44%
big-M	246.87	612.65	18.08%
big-M + 15 s timeout	290.21	15.00	3.75%
Extended	–	1800.00	–

We report aggregated results over 100 instances in Table 2. First, we are unable to solve even the LP relaxation of the extended method on any of the instances in the allotted 30 min, due to the quadratic growth in size. In contrast, the LP relaxation of the big-M + no cuts method can be solved very quickly. The big-M + (6b) method strengthens this LP bound by more than 15% on average, and only takes roughly 2.5× as long. This is not only because the separation runs very quickly, but also for a technical reason: when Gurobi's cutting planes are disabled, the callback separating over (6b) is only called a small number of times, as determined by Gurobi's internal cut selection procedure. Therefore, this 15% improvement is the result of only a small number of separation rounds, not an exhaustive iterative procedure (i.e. Gurobi terminates the cut loop well before all violated inequalities have been separated).

We may compare these results against the big-M method, which is able to provide a modestly better bound (roughly 18% improvement), but requires almost two orders of magnitude more time to produce the bound. For another comparison, *big-M + 15* s *timeout*, we set a smaller time limit of 15 s on Gurobi, which is a tighter upper bound on the maximum time used by the big-M + (6b) method. In this short amount of time, Gurobi is not able to improve the bound substantially, with less than 4% improvement. This suggests that the inequalities (6b) are not trivial to infer by generic cutting plane methods, and that it takes Gurobi many rounds of cut generation to achieve the same level of bound improvement we derive from restricting ourselves to those cuts in (6b).

Acknowledgement. The authors gratefully acknowledge Yeesian Ng and Ondřej Sýkora for many discussions on the topic of this paper, and for their work on the development of the `tf.opt` package used in the computational experiments.

A Deferred Proofs

A.1 Proof of Proposition 1

Proof. The result follows from applying Fourier–Motzkin elimination to (5a–5f) to project out the x^0, x^1, y^0, and y^1 variables; see [31, Chap. 13] for an explanation of the approach. We start by eliminating the x^1, y^0, and y^1 using the equations in (5a), (5b), and (5c), respectively, leaving only x^0.

First, if there is some input component i with $w_i = 0$, then x_i^0 only appears in the constraints (5d–5e), and so the elimination step produces $L_i \leqslant x_i \leqslant U_i$.

Second, if there is some i with $w_i < 0$, then we introduce an auxiliary variable $\tilde{x}_i$ with the equation $\tilde{x}_i = -x_i$. We then replace $w_i \leftarrow |w_i|$, $L_i \leftarrow -U_i$, and $U_i \leftarrow -L_i$, and proceed as follows under the assumption that $w > 0$.

Applying the Fourier-Motzkin procedure to eliminate x_1^0 gives the inequalities

$$\begin{aligned}
y &\geqslant w \cdot x + b \\
y &\leqslant w_1 x_1 - w_1 L_1 (1 - z) + \sum_{i>1} w_i x_i^0 + bz \\
y &\leqslant w_1 U_1 z + \sum_{i>1} w_i x_i^0 + bz \\
y &\geqslant w_1 x_1 - w_1 U_1 (1 - z) + \sum_{i>1} w_i x_i^0 + bz \\
y &\geqslant w_1 L_1 z + \sum_{i>1} w_i x_i^0 + bz \\
L_1 &\leqslant x_1 \leqslant U_1,
\end{aligned}$$

along with the existing inequalities in (5a–5f) where the x_1^0 coefficient is zero. Repeating this procedure for each remaining component of x^0 yields the linear system

$$y \geqslant w \cdot x + b \tag{7a}$$

$$y \leqslant \sum_{i \in I} w_i x_i - \sum_{i \in I} w_i L_i (1 - z) + \left(b + \sum_{i \notin I} w_i U_i \right) z \quad \forall I \subseteq \operatorname{supp}(w) \tag{7b}$$

$$y \geqslant \sum_{i \in I} w_i x_i - \sum_{i \in I} w_i U_i (1 - z) + \left(b + \sum_{i \notin I} w_i L_i \right) z \quad \forall I \subseteq \operatorname{supp}(w) \tag{7c}$$

$$(x, y, z) \in [L, U] \times \mathbb{R}_{\geqslant 0} \times [0, 1]. \tag{7d}$$

Moreover, we can show that the family of inequalities (7c) is redundant, and can therefore be removed. Fix some $I \subseteq \operatorname{supp}(w)$, and take $h(I) \stackrel{\text{def}}{=} \sum_{i \in I} w_i \breve{L}_i + \sum_{i \notin I} w_i \breve{U}_i + b$. If $h([\![\eta]\!] \setminus I) \geqslant 0$, we can express the inequality in (7c) corre-

sponding to the set I as a conic combination of the remaining constraints as:

$$\begin{aligned} y &\geqslant w \cdot x + b && \times \quad 1 \\ 0 &\geqslant L_i - x_i && \times \quad w_i && \forall i \notin I \\ 0 &\geqslant z - 1 && \times \quad h([\![\eta]\!] \setminus I) \end{aligned}$$

Alternatively, if $h([\![\eta]\!] \setminus I) < 0$, we can express the inequality in (7c) corresponding to the set I as a conic combination of the remaining constraints as:

$$\begin{aligned} y &\geqslant 0 && \times \quad 1 \\ 0 &\geqslant x_i - U_i && \times \quad w_i && \forall i \in I \\ 0 &\geqslant -z && \times \quad -h([\![\eta]\!] \setminus I) \end{aligned}$$

To complete the proof, for any components i where we introduced an auxiliary variable $\tilde{x}_i$, we use the corresponding equation $\tilde{x}_i = -x_i$ to eliminate x_i and replace it $\tilde{x}_i$, giving the result. □

A.2 Proof of Proposition 2

Proof. We fix $I = \{\kappa + 1, \ldots, \eta\}$ for some κ; this is without loss of generality by permuting the rows of the matrices presented below. Additionally, we presume that $w \geqslant 0$, which allows us to infer that $\check{L} = L$ and $\check{U} = U$. This is also without loss of generality by appropriately interchanging $+$ and $-$ in the definition of the $\tilde{p}^k$ below. In the following, references to (6b) are taken to be references to the inequality in (6b) corresponding to the subset I.

Take the two points $p^0 = (x, y, z) = (L, 0, 0)$ and $p^1 = (U, f(U), 1)$. Each point is feasible with respect to (6a–6c) and satisfies (6b) at equality. Then for some $\epsilon > 0$ and for each $i \in [\![\eta]\!] \setminus I$, take $\tilde{p}^i = (x, y, z) = (L + \epsilon \mathbf{e}^i, 0, 0)$. Similarly, for each $i \in I$, take $\tilde{p}^i = (x, y, z) = (U - \epsilon \mathbf{e}^i, f(U - \epsilon \mathbf{e}^i), 1)$. From the strict activity assumption, there exists some $\epsilon > 0$ sufficiently small such that each $\tilde{p}^k$ is feasible with respect to (6a–6c) and satisfies (6b) at equality.

This leaves us with $\eta + 2$ feasible points satisfying (6b) at equality; the result then follows by showing that the points are affinely independent. Take the matrix

$$\begin{pmatrix} p^1 - p^0 \\ \tilde{p}^1 - p^0 \\ \vdots \\ \tilde{p}^\kappa - p^0 \\ \tilde{p}^{\kappa+1} - p^0 \\ \vdots \\ \tilde{p}^\eta - p^0 \end{pmatrix} = \begin{pmatrix} U - L & f(U) & 1 \\ \epsilon \mathbf{e}^1 & 0 & 0 \\ \vdots & \vdots & \vdots \\ \epsilon \mathbf{e}^\kappa & 0 & 0 \\ U - L - \epsilon \mathbf{e}^{\kappa+1} & f(U - \epsilon \mathbf{e}^{\kappa+1}) & 1 \\ \vdots & \vdots & \vdots \\ U - L - \epsilon \mathbf{e}^\eta & f(U - \epsilon \mathbf{e}^\eta) & 1 \end{pmatrix} \cong \begin{pmatrix} U - L & f(U) & 1 \\ \epsilon \mathbf{e}^1 & 0 & 0 \\ \vdots & \vdots & \vdots \\ \epsilon \mathbf{e}^\kappa & 0 & 0 \\ -\epsilon \mathbf{e}^{\kappa+1} & -w_{\kappa+1}\epsilon & 0 \\ \vdots & \vdots & \vdots \\ -\epsilon \mathbf{e}^\eta & -w_\eta \epsilon & 0 \end{pmatrix},$$

where the third matrix is constructed by subtracting the first row to each of row $\kappa + 2$ to $\eta + 1$ (i.e. those corresponding to $\tilde{p}^i - p^0$ for $i > \kappa$), and is taken to mean congruency with respect to elementary row operations. If we permute

the last column (corresponding to the z variable) to the first column, we observe that the resulting matrix is upper triangular with a nonzero diagonal, and so has full row rank. Therefore, the starting matrix also has full row rank, as we only applied elementary row operations, and therefore the $\eta + 2$ points are affinely independent, giving the result. □

References

1. Amos, B., Xu, L., Kolter, J.Z.: Input convex neural networks. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 146–155. PMLR, International Convention Centre, Sydney, Australia, 06–11 August 2017
2. Anderson, R., Huchette, J., Tjandraatmadja, C., Vielma, J.P.: Strong convex relaxations and mixed-integer programming formulations for trained neural networks (2018). https://arxiv.org/abs/1811.01988
3. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: a brief survey. IEEE Signal Process. Mag. **34**(6), 26–38 (2017)
4. Atamtürk, A., Gómez, A.: Strong formulations for quadratic optimization with M-matrices and indicator variables. Math. Program. **170**, 141–176 (2018)
5. Balas, E.: Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. SIAM J. Algorithmic Discret. Methods **6**(3), 466–486 (1985)
6. Balas, E.: Disjunctive programming: properties of the convex hull of feasible points. Discret. Appl. Math. **89**, 3–44 (1998)
7. Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Neuron constraints to model complex real-world problems. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 115–129. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23786-7_11
8. Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Optimization and controlled systems: a case study on thermal aware workload dispatching. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, pp. 427–433 (2012)
9. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A.V., Criminisi, A.: Measuring neural net robustness with constraints. In: Advances in Neural Information Processing Systems, pp. 2613–2621 (2016)
10. Belotti, P., et al.: On handling indicator constraints in mixed integer programming. Comput. Optim. Appl. **65**(3), 545–566 (2016)
11. Bertsimas, D., Kallus, N.: From predictive to prescriptive analytics. Management Science (2018). https://arxiv.org/abs/1402.5481
12. Biggs, M., Hariss, R.: Optimizing objective functions determined from random forests (2017). https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2986630
13. Bonami, P., Lodi, A., Tramontani, A., Wiese, S.: On mathematical programming with indicator constraints. Math. Program. **151**(1), 191–223 (2015)
14. Bunel, R., Turkaslan, I., Torr, P.H., Kohli, P., Kumar, M.P.: A unified view of piecewise linear neural network verification. In: Advances in Neural Information Processing Systems (2018)
15. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57 (2017)
16. Cheng, C.-H., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 251–268. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_18

17. Deng, Y., Liu, J., Sen, S.: Coalescing data and decision sciences for analytics. In: Recent Advances in Optimization and Modeling of Contemporary Problems. INFORMS (2018)
18. Donti, P., Amos, B., Kolter, J.Z.: Task-based end-to-end model learning in stochastic optimization. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems, vol. 30, pp. 5484–5494. Curran Associates, Inc. (2017)
19. Dulac-Arnold, G., et al.: Deep reinforcement learning in large discrete action spaces (2015). https://arxiv.org/abs/1512.07679
20. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: Dutle, A., Muñoz, C., Narkawicz, A. (eds.) NFM 2018. LNCS, vol. 10811, pp. 121–138. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77935-5_9
21. Dvijotham, K., et al.:: Training verified learners with learned verifiers (2018). https://arxiv.org/abs/1805.10265
22. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T., Kohli, P.: A dual approach to scalable verification of deep networks. In: Thirty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (2018)
23. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
24. Elmachtoub, A.N., Grigas, P.: Smart "Predict, then Optimize" (2017). https://arxiv.org/abs/1710.08005
25. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints **23**, 296–309 (2018)
26. Gatys, L.A., Ecker, A.S., Bethge, M.: A neural algorithm of artistic style (2015). https://arxiv.org/abs/1508.06576
27. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning, vol. 1. MIT Press, Cambridge (2016)
28. den Hertog, D., Postek, K.: Bridging the gap between predictive and prescriptive analytics - new optimization methodology needed (2016). http://www.optimization-online.org/DB_HTML/2016/12/5779.html
29. Hijazi, H., Bonami, P., Cornuéjols, G., Ouorou, A.: Mixed-integer nonlinear programs featuring "on/off" constraints. Comput. Optim. Appl. **52**(2), 537–558 (2012)
30. Hijazi, H., Bonami, P., Ouorou, A.: A note on linear on/off constraints (2014). http://www.optimization-online.org/DB_FILE/2014/04/4309.pdf
31. Hooker, J.: Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction. Wiley, Hoboken (2011)
32. Huchette, J.: Advanced mixed-integer programming formulations: methodology, computation, and application. Ph.D. thesis, Massachusetts Institute of Technology (June 2018)
33. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
34. Khalil, E.B., Gupta, A., Dilkina, B.: Combinatorial attacks on binarized neural networks. In: International Conference on Learning Representations (2019)
35. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)

36. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**, 2278–2324 (1998)
37. Lombardi, M., Gualandi, S.: A lagrangian propagator for artificial neural networks in constraint programming. Constraints **21**(4), 435–462 (2016)
38. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward ReLU neural networks (2017). https://arxiv.org/abs/1706.07351
39. Mišić, V.V.: Optimization of tree ensembles (2017). https://arxiv.org/abs/1705.10883
40. Mladenov, M., Boutilier, C., Schuurmans, D., Elidan, G., Meshi, O., Lu, T.: Approximate linear programming for logistic Markov decision processes. In: Proceedings of the Twenty-sixth International Joint Conference on Artificial Intelligence (IJCAI 2017), pp. 2486–2493, Melbourne, Australia (2017)
41. Mordvintsev, A., Olah, C., Tyka, M.: Inceptionism: going deeper into neural networks (2015). https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html
42. Olah, C., Mordvintsev, A., Schubert, L.: Feature Visualization. Distill (2017). https://distill.pub/2017/feature-visualization
43. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: IEEE European Symposium on Security and Privacy, pp. 372–387, March 2016
44. Say, B., Wu, G., Zhou, Y.Q., Sanner, S.: Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, pp. 750–756 (2017)
45. Schweidtmann, A.M., Mitsos, A.: Global deterministic optimization with artificial neural networks embedded. J. Optim. Theory Appl. **180**(3), 925–948 (2019)
46. Serra, T., Ramalingam, S.: Empirical bounds on linear regions of deep rectifier networks (2018). https://arxiv.org/abs/1810.03370
47. Serra, T., Tjandraatmadja, C., Ramalingam, S.: Bounding and counting linear regions of deep neural networks. In: Thirty-Fifth International Conference on Machine Learning (2018)
48. Szegedy, C., et al.: Intriguing properties of neural networks. In: International Conference on Learning Representations (2014)
49. Tjeng, V., Xiao, K., Tedrake, R.: Verifying neural networks with mixed integer programming. In: International Conference on Learning Representations (2019)
50. Vielma, J.P.: Mixed integer linear programming formulation techniques. SIAM Rev. **57**(1), 3–57 (2015)
51. Vielma, J.P.: Small and strong formulations for unions of convex sets from the Cayley embedding. Math. Program. (2018)
52. Vielma, J.P., Nemhauser, G.: Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. Math. Program. **128**(1–2), 49–72 (2011)
53. Wong, E., Kolter, J.Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: International Conference on Machine Learning (2018)
54. Wong, E., Schmidt, F., Metzen, J.H., Kolter, J.Z.: Scaling provable adversarial defenses. In: 32nd Conference on Neural Information Processing Systems (2018)

55. Wu, G., Say, B., Sanner, S.: Scalable planning with Tensorflow for hybrid nonlinear domains. In: Advances in Neural Information Processing Systems, pp. 6276–6286 (2017)
56. Xiao, K.Y., Tjeng, V., Shafiullah, N.M., Madry, A.: Training for faster adversarial robustness verification via inducing ReLU stability. In: International Conference on Learning Representations (2019)