Optimization Over Trained Neural Networks: Taking a Relaxing Walk

Jiatai Tong[1], Junyang Cai[1,2], and Thiago Serra[1]

[1] Bucknell University, Lewisburg PA, United States
{jt037,jc092,thiago.serra}@bucknell.edu
[2] University of Southern California, Los Angeles CA, United States
caijunya@usc.edu

Abstract. Besides training, mathematical optimization is also used in deep learning to model and solve formulations over trained neural networks for purposes such as verification, compression, and optimization with learned constraints. However, solving these formulations soon becomes difficult as the network size grows due to the weak linear relaxation and dense constraint matrix. We have seen improvements in recent years with cutting plane algorithms, reformulations, and an heuristic based on Mixed-Integer Linear Programming (MILP). In this work, we propose a more scalable heuristic based on exploring global and local linear relaxations of the neural network model. Our heuristic is competitive with a state-of-the-art MILP solver and the prior heuristic while producing better solutions with increases in input, depth, and number of neurons.

Keywords: Deep Learning · Mixed-Integer Linear Programming · Linear Regions · Neural Surrogate Models · Rectified Linear Units.

1 Introduction

There is a natural role for mathematical optimization in machine learning with training, where discrete optimization has a "discreet" but growing presence in classification trees [13, 12, 82, 83, 45, 81, 89, 37, 21, 2, 28, 3], decision diagrams [44, 34], decision rules [1, 51], and neural networks [47, 50, 68, 64, 14, 77, 11, 7].

Now a new role has emerged with predictions from machine learning models being used as part of the formulation of optimization problems. For example, imagine that we train a neural network on historical data for approximating an objective function that we are not able to represent explicitly. Overall, we start with one or more trained machine learning models, and then we formulate an optimization model which—among other things—represents the relationship between decision variables for the inputs and outputs of those trained models. Since other discrete decision variables and constraints may be part of such optimization models, gradient descent is not as convenient here as it is for training.

These formulations are *neural surrogate models* if involving neural networks. Neural networks are essentially nonlinear, and thus challenging to model in mathematical optimization. However, we can use Mixed-Integer Linear Programming (MILP) for popular activations such as the Rectified Linear Unit (ReLU)

[39, 61, 36, 53, 67]. Neural networks with ReLUs represent piecewise linear functions [63, 6], which we model in MILP with binary decision variables altering the slopes [72]. As with other activations [26, 35, 43], ReLU networks have been shown to be universal function approximators with one hidden layer but enough neurons [88] and with limited neurons per layer but enough layers [55, 42, 62].

Many frameworks to formulate neural surrogate models have emerged—JANOS [10], OMLT [22], OCL [32], OptiCL [56], and Gurobi Machine Learning [38]—in addition to stochastic and robust optimization variants [30, 29, 49]. The applications in machine learning include network verification [24, 4, 5, 69, 75], network pruning [74, 73, 31], counterfactual explanation [48], and constrained reinforcement learning [27, 18]. In the broader line of work often denoted as *constraint learning*, these models have been used for scholarship allocation [10], patient survival in chemotherapy [56], power generation [60] and voltage regulation [23] in power grids, boiling point optimization in molecular design [57], and automated control of industrial operations in general [70, 85, 87].

However, these models can be difficult to solve as they grow in size. They have weak linear relaxations due to the dense constraint matrix within each layer and the big M constraints for each neuron, which sparked immediate and continued interest in calibrating big M coefficients [24, 33, 54, 80, 8] as well as in strengthening the formulation and generating cutting planes [4, 5, 80]. Other improvements include identifying stable neurons [78, 86], exploiting the dependency among neural activations [16, 71], and inducing sparser formulations by network pruning [70, 19]. But at the rate of one binary variable per ReLU, typically-sized neural networks entail considerably large MILPs, hence limiting the applications where these models are solvable within reasonable time.

We may expect that improving scalability will require algorithms exploiting the model structure. For example, Fischetti and Jo [33] first observed that a feasible solution for the MILP mapping from inputs to outputs of a single neural network is immediate once a given input is chosen. This strategy has been shown effective at least twice [73, 65], whereas finding a feasible solution for an MILP is generally NP-complete [25]. Another example of special structure comes from ReLU networks representing piecewise linear functions. Within each part of the domain mapped as a linear function, which is denoted as a *linear region*, there is a direction for locally improving the output. In fact, Perakis and Tsiourvas [65] developed a local search heuristic that moves along adjacent linear regions by solving restrictions of the MILP model with some binary variables fixed. However, the reliance on MILP eventually brings scalability issues back—although much later in comparison to solving the model without restrictions.

But is there hope for optimization over linear regions at scale? That is akin to thinking about MILPs as unions of polyhedra in disjunctive programming [9]. While earlier studies have shown that the number of linear regions may grow fast on model dimensions [63, 59, 76, 66, 6], later studies have shown that there are architectureal tradeoffs limiting such growth [58, 72, 71, 20]. Moreover, the networks with typical distributions of parameters have considerably fewer linear regions [40, 41]; and gradients change little between adjacent linear regions [84].

Hence, we may conjecture that the search space is actually smaller and simpler than expected, and thus that a leaner algorithm may produce good results faster.

In this work, we propose an heuristic based on solving a Linear Programming (LP) model rather than an MILP model at each step of the local search, and we generate initial solutions with LP relaxations of the neural surrogate model. Confirming our intuition, this strategy is computationally better at scale, such as when neural networks have larger inputs, more neurons, or greater depths.

2 Notation and Conventions

In this paper, we consider feedforward networks with fully-connected layers of neurons having ReLU activation. Note that convolutional layers can be represented as fully-connected layers with a block-diagonal weight matrix. We also abstract that fully-connected layers are often followed by a softmax layer [17], since the largest input of softmax matches the largest output of softmax.

We assume that the neural network has an input $\boldsymbol{x} = [x_1 \ x_2 \ \dots \ x_{n_0}]^\top$ from a bounded domain $\mathbb{X}$ and corresponding output $\boldsymbol{y} = [y_1 \ y_2 \ \dots \ y_m]^\top$, and each layer $l \in \mathbb{L} = \{1, 2, \dots, L\}$ has output $\boldsymbol{h}^l = [h_1^l \ h_2^l \dots h_{n_l}^l]^\top$ from neurons indexed by $i \in \mathbb{N}_l = \{1, 2, \dots, n_l\}$. Let $\boldsymbol{W}^l$ be the $n_l \times n_{l-1}$ matrix where each row corresponds to the weights of a neuron of layer l, $\boldsymbol{W}_i^l$ the i-th row of $\boldsymbol{W}^l$, and $\boldsymbol{b}^l$ the vector of biases associated with the units in layer l. With $\boldsymbol{h}^0$ for $\boldsymbol{x}$ and $\boldsymbol{h}^L$ for $\boldsymbol{y}$, the output of each unit i in layer l consists of an affine function $g_i^l = \boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l$ followed by the ReLU activation $h_i^l = \max\{0, g_i^l\}$. We denote the neuron *active* when $h_i^l = g_i^l > 0$ and *inactive* when $h_i^l = 0$ and $g_i^l < 0$. When $h_i^l = g_i^l = 0$, the state is given by the last nonzero value of g_i^l during local search.

In typical neural surrogate models, the parameters $\boldsymbol{W}^l$ and $\boldsymbol{b}^l$ of each layer $l \in \mathbb{L}$ are constant. The decision variables are the inputs of the network ($\boldsymbol{x} = \boldsymbol{h}^0 \in \mathbb{X}$) and, in each layer, the outputs before and after activation ($\boldsymbol{g}^l \in \mathbb{R}^{n_l}$ and $\boldsymbol{h}^l \in \mathbb{R}_+^{n_l}$ for $l \in \mathbb{L}$) as well as the activation states ($\boldsymbol{z}^l \in \{0,1\}^{n_l}$ for $l \in \mathbb{L}$). By linearly mapping these variables according to the parameters of the network, each possible combination of inputs, outputs, and activations become a solution of an MILP formulation. For each layer $l \in \mathbb{L}$ and neuron $i \in \mathbb{N}_l$, the following constraints associate its decision variables $\boldsymbol{h}^l$, $\boldsymbol{g}_i^l$, $\boldsymbol{h}_i^l$, and $\boldsymbol{z}_i^l$:

$$\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l = \boldsymbol{g}_i^l \tag{1}$$

$$(\boldsymbol{z}_i^l = 1) \to \boldsymbol{h}_i^l = \boldsymbol{g}_i^l \tag{2}$$

$$(\boldsymbol{z}_i^l = 0) \to (\boldsymbol{g}_i^l \leq 0 \wedge \boldsymbol{h}_i^l = 0) \tag{3}$$

$$\boldsymbol{h}_i^l \geq 0 \tag{4}$$

$$\boldsymbol{z}_i^l \in \{0, 1\} \tag{5}$$

The indicator constraints (2)–(3) can be modeled with big M constraints [15].

We follow the convention of characterizing each linear region by the set of neurons that they activate [66]. For an input $\boldsymbol{x}$, let $\mathbb{S}^l(\boldsymbol{x}) \subseteq \{1, 2, \dots, n_l\}$ denote the *activation set* of layer l. Hence, layer l defines an affine transformation of the form $\Omega^{\mathbb{S}^l(\boldsymbol{x})}(\boldsymbol{W}^l \boldsymbol{h}^{l-1} + \boldsymbol{b}^l)$, where $\Omega^{\mathbb{U}}$ is a diagonal $v \times v$ matrix

Algorithm 1 Local search to walk within and across linear regions.

```
repeat                                                          ▷ Local search consists of an improvement loop
    $\boldsymbol{x}^1 \leftarrow$ Optimal solution of $\mathbf{LP}(\boldsymbol{x}^0)$    ▷ Finds best solution within linear region
    if $F(\boldsymbol{x}^1) > F(\boldsymbol{x}^0)$ then                             ▷ Checks if there was an improvement
        $\boldsymbol{d} \leftarrow \boldsymbol{x}^1 - \boldsymbol{x}^0$                                   ▷ Computes direction of improvement $\boldsymbol{d}$
        $\boldsymbol{x}^0 \leftarrow \boldsymbol{x}^1 + \varepsilon \boldsymbol{d}$                           ▷ Leaves the linear region along direction $\boldsymbol{d}$
        for $i \leftarrow 1, \dots, n_0$ do                             ▷ Loops over all input dimensions
            if $x^1 + \varepsilon \boldsymbol{d} e^i \notin \mathbb{X}$ then                  ▷ Checks if move is outside input space
                $x_i^0 \leftarrow x_i^1$                                 ▷ Corrects move to be inside input space
            end if
        end for
    end if
until $F(\boldsymbol{x}^1) \leq F(\boldsymbol{x}^0)$                                ▷ Stops when no improvement occurs
return $\boldsymbol{x}^0$                                                 ▷ Returns best solution found
```

in which $\Omega_{ii}^{\mathbb{U}} = 1$ if $i \in \mathbb{U}$ and $\Omega_{ii}^{\mathbb{U}} = 0$ otherwise for a subset $\mathbb{U} \subseteq \mathbb{V} = \{1, 2, \dots, v\}$. For the linear region containing $\boldsymbol{x} = \boldsymbol{x}^0$, the output of the neural network is the affine transformation $\boldsymbol{T}\boldsymbol{t} + \boldsymbol{t}$ for $\boldsymbol{T} = \prod_{l=1}^{L} \Omega^{\mathbb{S}^l(\boldsymbol{x}^0)} \boldsymbol{W}^l$ and $\boldsymbol{t} = \sum_{l'=1}^{L} \left(\prod_{l''=l'+1}^{L} \Omega^{\mathbb{S}^{l''}(\boldsymbol{x}^0)} \boldsymbol{W}^{l''} \right) \Omega^{\mathbb{S}^{l'}(\boldsymbol{x}^0)} \boldsymbol{b}^{l'}$, in comparison to which we note that the output h^ℓ of layer ℓ is obtained by replacing L with ℓ [46].

3 Walking Along Linear Regions

Let us consider a neural network representing the piecewise linear function $f(\boldsymbol{x})$, a linear objective function $F(\boldsymbol{x}) = \boldsymbol{c}^\top f(\boldsymbol{x})$ to be maximized, and an implicit set of linear constraints from assuming the input set $\mathbb{X}$ to be a polytope.

We can model our problem as an MILP on $\left(\boldsymbol{x}, \{\boldsymbol{g}\}_{i=1}^{L}, \{\boldsymbol{h}\}_{i=0}^{L}, \{\boldsymbol{z}\}_{i=1}^{L}, \boldsymbol{y}\right)$:

$$\max \; \boldsymbol{c}^\top \boldsymbol{y} \tag{6}$$

$$\text{s.t. } (1)\text{–}(5) \qquad \forall l \in \mathbb{L}, i \in \{1, \dots, n_l\} \tag{7}$$

$$\boldsymbol{x} = \boldsymbol{h}^0, \boldsymbol{y} = \boldsymbol{h}^L, \boldsymbol{x} \in \mathbb{X} \tag{8}$$

For an input $\boldsymbol{x} = \boldsymbol{x}^0$, we can define an LP model by fixing the binary variables as $\boldsymbol{z}_i^l = 1$ if $i \in \mathbb{S}^\ell(\boldsymbol{x}^0)$ and $\boldsymbol{z}_i^l = 0$ otherwise. Let us denote it as $\mathbf{LP}(\boldsymbol{x}^0)$. By not fixing $\boldsymbol{x}$, $\mathbf{LP}(\boldsymbol{x}^0)$ finds an input maximizing $F(\boldsymbol{x})$ in a linear region with $\boldsymbol{x}^0$.

We propose the local search outlined in Algorithm 1, which is a loop moving from an input $\boldsymbol{x}^0$ to the input $\boldsymbol{x}^1$ in the same linear region by solving $\text{LP}(\boldsymbol{x}^0)$. If we find an improvement, we continue moving along the same direction $\boldsymbol{d} = \boldsymbol{x}^1 - \boldsymbol{x}^0$ to the next linear region with a step $\varepsilon \boldsymbol{d}$ updating $\boldsymbol{x}^0$. We expect that $F(\boldsymbol{x}^1 + \varepsilon \boldsymbol{d}) > F(\boldsymbol{x}^1)$ since $\|\nabla F(\boldsymbol{x}^1 + \varepsilon \boldsymbol{d}) - \nabla F(\boldsymbol{x}^1)\|$ is usually small [84]. Ideally, ε should be small enough to move only to the next linear region while being large enough to be numerically computed as in the relative interior of that next linear region. We also adjust the move along each dimension to ensure that $\boldsymbol{x}^0 \in \mathbb{X}$.

Figure 1 illustrates three iterations of improvement with the local search algorithm, each characterized by a pair of points $(\boldsymbol{x}^0, \boldsymbol{x}^1)$ denoting a direction of improvement: (A, B), (C, D), and (E, F). Among those, the second iteration shows that a larger step may skip a smaller linear region. Conversely, we could mistakenly conclude that no further improvement is possible if a smaller step $\boldsymbol{d}$ is numerically computed in such a way that $\boldsymbol{x}^1 = H \approxeq \boldsymbol{x}^1 + \varepsilon \boldsymbol{d}$.

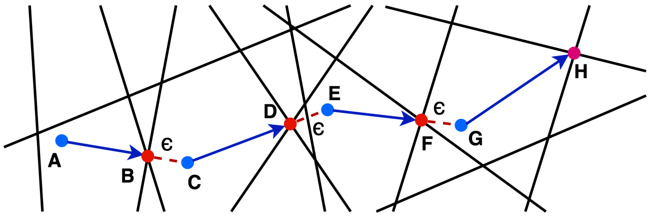

Fig. 1. From a starting point, our local search algorithm moves in a certain direction indicated by the blue arrow, and then takes a small step into the next linear region before moving again. We stop when the next linear region has no better solution.

We embed the local search in a generator of initial solution outlined in Algorithm 2, which is based on solving variations of the linear relaxation of (6)–(8). We can compute an input $\tilde{x}$ that is somewhat aligned with maximizing function $F(\boldsymbol{x})$ by solving this relaxation. We denote this model as LR:

$$\max \ \boldsymbol{c}^\top \boldsymbol{y} \tag{9}$$

$$\text{s.t. } (1)\text{–}(4) \qquad \forall l \in \mathbb{L}, i \in \{1, \ldots, n_l\} \tag{10}$$

$$\boldsymbol{z}_i^l \in [0, 1] \qquad \forall l \in \mathbb{L}, i \in \{1, \ldots, n_l\} \tag{11}$$

$$\boldsymbol{x} = \boldsymbol{h}^0, \boldsymbol{y} = \boldsymbol{h}^L, \boldsymbol{x} \in \mathbb{X} \tag{12}$$

We then impose a random sequence of constraints on activation states, producing a solution $\bar{x}$ in a different linear region after adding each constraint. The probability of fixing a neuron are calibrated to produce the most change to the linear relaxation. We start over from $\tilde{x}$ when no more activations can be fixed.

Related Work We denote our approach as Relax-and-Walk (**RW**) and the local search in [65] as "Sample-and-MIP" (**SM**). SM is based on generating initial solutions by random sampling and then solving a restriction of the MILP (6)–(8) to identify the best solution among adjacent linear regions. SM may find the best solution locally, but it may take much longer to compute in networks with larger dimensions. Hence, SM may produce fewer solutions and less improvement.

4 Experiments

We benchmark our **RW** method with **SM** [65] and **Gurobi** 10.0.1. For local search, we use $\varepsilon = 0.01$ since by preliminary tests it was small enough to avoid skipping linear regions. We ran the code in [79] on 10 cores of a cluster with Intel(R) Xeon(R) Gold 6336Y CPU @ 2.40GHz processors and 16 GB of RAM.

Algorithm 2 Generation of initial solutions and injection in local search

```
$(\widetilde{\boldsymbol{x}}, \{\widetilde{\boldsymbol{g}}\}_{i=1}^{L}, \{\widetilde{\boldsymbol{h}}\}_{i=0}^{L}, \{\widetilde{\boldsymbol{z}}\}_{i=1}^{L}, \widetilde{\boldsymbol{y}}) \leftarrow$ Optimal solution of LR
DO LOCAL SEARCH FROM $\widetilde{\boldsymbol{x}}$   ▷ First local search; and the only one at $\widetilde{\boldsymbol{x}}$
loop   ▷ Outer loop defines indefinite run until interrupted
    $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{z}}) \leftarrow (\widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{z}})$   ▷ Uses first LR solution to decide where to go next
    for $\ell \leftarrow 1, \dots, L$ do   ▷ Loops sequentially over layers to fix neurons
        $\mathbb{N} \leftarrow \{1, 2, \dots, n_\ell\}$   ▷ Accounts for all neurons to fix in layer $\ell$
        while $\mathbb{N} \neq \emptyset$ do   ▷ Loops to try fixing each neuron once
            for $i \in N$ do   ▷ Loops over unfixed neurons
                if $i \in \mathbb{S}^{\ell}(\bar{\boldsymbol{x}})$ then   ▷ Checks if neuron $i$ is active in last solution $\bar{x}$
                    $\chi_i \leftarrow 1 - \bar{\boldsymbol{z}}_i^{\ell}$   ▷ If so, measures distance of relaxed binary to 1
                else
                    $\chi_i \leftarrow \bar{\boldsymbol{z}}_i^{\ell}$   ▷ Otherwise, measures distance of relaxed binary to 0
                end if
            end for   ▷ Produces a shifted probability on $\chi$ values to pick a neuron
            $k \leftarrow$ Element $i \in \mathbb{N}$ with probability $\chi_i + \delta / \sum_{j \in \mathbb{N}} (\chi_j + \delta)$
            $\mathbb{N} \leftarrow \mathbb{N} \setminus \{k\}$   ▷ Records attempt to fix neuron $k \in \mathbb{N}$
            if $k \in \mathbb{S}^{\ell}(\bar{x})$ then   ▷ Checks if neuron $k$ is active in last solution $\bar{x}$
                Add constraint $\boldsymbol{z}_k^{\ell} = 0$ to LR ▷ If so, makes it inactive going forward
            else
                Add constraint $\boldsymbol{z}_k^{\ell} = 1$ to LR   ▷ Otherwise, makes it active
            end if
            if LR is feasible then   ▷ Checks if new constraint keeps LR feasible
                $\big(\bar{\boldsymbol{x}}, \{\bar{\boldsymbol{g}}\}_{i=1}^{L}, \{\bar{\boldsymbol{h}}\}_{i=0}^{L}, \{\bar{\boldsymbol{z}}\}_{i=1}^{L}, \bar{\boldsymbol{y}}\big) \leftarrow$ Optimal solution of LR
                DO LOCAL SEARCH FROM $\bar{\boldsymbol{x}}$ ▷ Local search at new solution
            else   ▷ In case not, neuron can only have same activation as before
                Remove constraint on $\boldsymbol{z}_k^{\ell}$; revert $(\bar{x}, \bar{z})$ to last feasible solution of LR
            end if
        end while   ▷ Fixed the entire layer; moves on to the next
    end for   ▷ Fixed all layers; ready to drop constraints
    Remove all activation constraints from LR
end loop   ▷ Starts over from $(\widetilde{x}, \widetilde{z})$
```

4.1 Random ReLU Networks

Our first experiment replicates and extends the optimization of output value of randomly initialized neural networks in [65] to test scalability and solution quality. With a time limit of 1 hour, we use 5 different networks for each choice of input sizes $n_0 \in \{10, 100, 1000\}$ and configurations of the form $L \times n_\ell$ for depth $L \in \{1, 2, 3\}$ and width $n_\ell \in \{100, 500\}$. We note that solving to optimality with Gurobi within 1 hour is very unlikely, except for 1×100 with $n_0 = 10$.

RW vs. SM: Figure 2 shows the pair of values obtained for the same random network with RW and SM. RW outperforms SM for $n_0 \in \{100, 1000\}$ and $n_\ell = 500$. The performance is similar for $n_0 = 10$, except in the four cases where Gurobi fails to solve the linear relaxation. Those have minimum value in the plots. That happens more often when n_0 is the smallest while L is larger: the model is likely more sensitive to numerical issues as the linear regions get smaller.

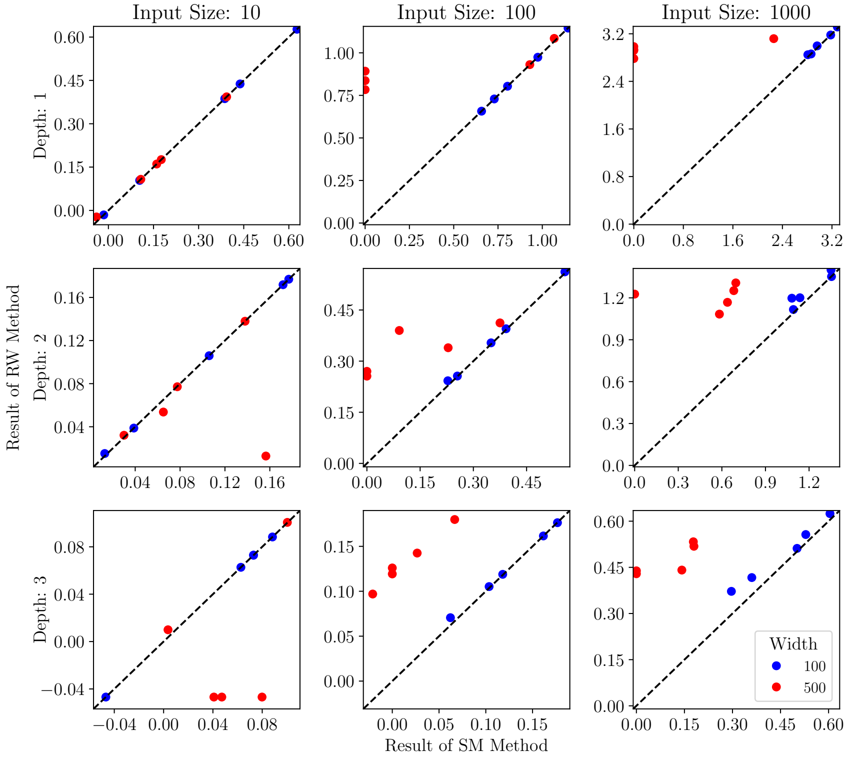

Fig. 2. Comparison of best objective values obtained by RW and SM in random networks. The points are favorable to RW above the line $Y = X$; and to SW below it. RW is at least 1% better in 64.4% of the cases, while SM is in 12.2%.

Table 1. Average number of final solutions produced by local search with each method.

Model Size	$n_0 = 10$		$n_0 = 100$		$n_0 = 1000$	
	RW	SM	RW	SM	RW	SM
1 × 100	2882.0	2562.2	1038.4	548.0	134.6	100.2
1 × 500	174.8	165.4	80.0	4.4	16.2	1.0
2 × 100	429.2	230.4	331.0	39.8	55.8	11.8
2 × 500	10.0	2.8	25.4	1.0	6.0	1.0
3 × 100	421.8	197.0	204.0	18.2	33.8	4.2
3 × 500	11.5	2.0	15.0	1.0	3.0	1.0

Moreover, we observe that **walking is cheaper than MIPing**: Table 1 shows that the walking algorithm RW converges more frequent to a local optimum by the time limit. Conversely, the average runtime of MILP restrictions in SM explodes very quickly when the network gets wider, consistent with Figure 2. This can be explained by the number of unfixed MILP variables growing with the network dimensions in the SM approach. Moreover, consecutive steps of the SM approach may reevaluate some neighboring linear regions again.

RW vs. Gurobi: Figure 3 shows a similar comparison between RW and Gurobi, but with depths combined for conciseness. Gurobi can handle a shallow network ($L = 1$) even with the largest input size $n_0 = 1000$. When the network is deeper and the structure of the linear regions more complex [72], directly solving an MILP is slower. When both width and depth are large, Gurobi cannot find a feasible solution—see points next to Y-axis. The same four cases with unbounded relaxation are also difficult for Gurobi—see points near the left bottom.

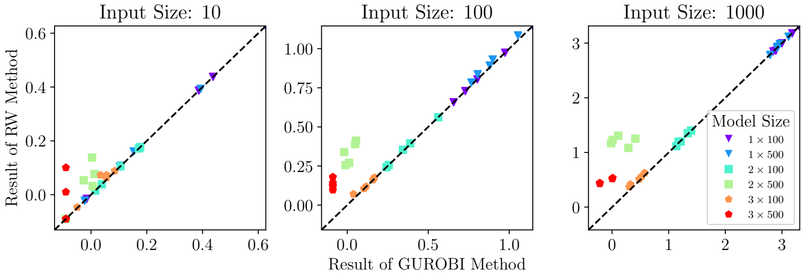

Fig. 3. Comparison of best objective values obtained by RW and Gurobi in random networks. RW is at least 1% better in 55.1% of the cases, while Gurobi is in 12.4%.

4.2 Optimal Adversary Experiment

Given an input $\boldsymbol{x} = \hat{\boldsymbol{x}}$, the output neuron for the predicted label c, and the output neuron for another likely label w, the optimal adversary problem aims to maximize $\boldsymbol{y}_w - \boldsymbol{y}_c$ for an input $\boldsymbol{x}$ sufficiently near $\hat{\boldsymbol{x}}$. We solved this problem for $|\boldsymbol{x} - \hat{\boldsymbol{x}}|_1 < \Delta$ as in [80] by using a setup derived from the Gurobi Machine Learning repository [38]. Figure 4 shows the result from testing 50 images from the MNIST dataset [52] with $\Delta = 5$ for 1 hour, all of which on a 2×500 classifier with test accuracy 97.04%. In 10% of the cases, RW found an adversarial input (positive solution) and Gurobi did not. When RW does better, it does so by a wider margin.

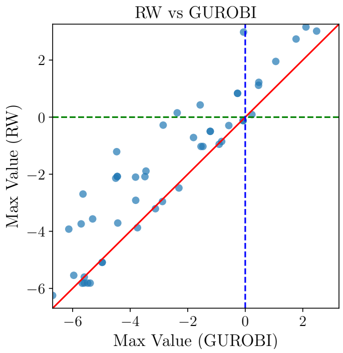

Fig. 4. Comparison of best objective values obtained by RW and Gurobi in optimal adversary models. RW is at least 1% better in 68% of the cases, while Gurobi is in 30%.

5 Conclusion

We introduced a local search algorithm for optimizing over trained neural networks. We designed our algorithm to leverage model structure based on what is known about linear regions in deep learning. Moreover, our algorithm scales more easily because it only solves LP models at every step. Last, but certainly not least, the solutions are usually better in comparison with other methods.

Acknowledgement The authors were supported by the National Science Foundation (NSF) grant IIS 2104583, including Junyang Cai while at Bucknell.

References

1. Boolean decision rules via column generation. Neural Information Processing Systems (NeurIPS) (2018)
2. Aghaei, S., Gómez, A., Vayanos, P.: Strong optimal classification trees. arXiv:2103.15965 (2021)
3. Alston, B., Validi, H., Hicks, I.V.: Mixed integer linear optimization formulations for learning optimal binary classification trees. arXiv:2206.04857 (2022)
4. Anderson, R., Huchette, J., Tjandraatmadja, C., Vielma, J.: Strong mixed-integer programming formulations for trained neural networks. In: Integer Programming and Combinatorial Optimization (IPCO) (2019)
5. Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P.: Strong mixed-integer programming formulations for trained neural networks. Mathematical Programming (2020)
6. Arora, R., Basu, A., Mianjy, P., Mukherjee, A.: Understanding deep neural networks with rectified linear units. In: International Conference on Learning Representations (ICLR) (2018)
7. Aspman, J., Korpas, G., Marecek, J.: Taming binarized neural networks and mixed-integer programs. arXiv:2310.04469 (2023)
8. Badilla, F., Goycoolea, M., Muñoz, G., Serra, T.: Computational tradeoffs of optimization-based bound tightening in ReLU networks (2023)
9. Balas, E.: Disjunctive Programming. Springer Cham (2018)
10. Bergman, D., Huang, T., Brooks, P., Lodi, A., Raghunathan, A.U.: JANOS: An integrated predictive and prescriptive modeling framework. INFORMS Journal on Computing (2022)
11. Bernardelli, A.M., Gualandi, S., Lau, H.C., Milanesi, S.: The BeMi stardust: A structured ensemble of binarized neural networks. In: Learning and Intelligent Optimization (LION) (2023)
12. Bertsimas, D., Dunn, J.: Optimal classification trees. Machine Learning (2017)
13. Bertsimas, D., Shioda, R.: Classification and regression via integer optimization. Operations Research (2007)
14. Bienstock, D., Muñoz, G., Pokutta, S.: Principled deep neural network training through linear programming. Discrete Optimization (2023)
15. Bonami, P., Lodi, A., Tramontani, A., Wiese, S.: On mathematical programming with indicator constraints. Mathematical Programming (2015)
16. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of relu-based neural networks via dependency analysis. In: AAAI Conference on Artificial Intelligence (AAAI) (2020)
17. Bridle, J.S.: Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In: Neurocomputing, pp. 227–236 (1990)
18. Burtea, R.A., Tsay, C.: Safe deployment of reinforcement learning using deterministic optimization over neural networks. In: Computer Aided Chemical Engineering, vol. 52, pp. 1643–1648. Elsevier (2023)
19. Cacciola, M., Frangioni, A., Lodi, A.: Structured pruning of neural networks for constraints learning. arXiv:2307.07457 (2023)

20. Cai, J., Nguyen, K.N., Shrestha, N., Good, A., Tu, R., Yu, X., Zhe, S., Serra, T.: Getting away with more network pruning: From sparsity to geometry and linear regions. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR) (2023)
21. Carrizosa, E., Molero-Río, C., Morales, D.R.: Mathematical optimization in classification and regression trees. TOP (2021)
22. Ceccon, F., Jalving, J., Haddad, J., Thebelt, A., Tsay, C., Laird, C.D., Misener, R.: Omlt: Optimization & machine learning toolkit. Journal of Machine Learning Research **23**(349), 1–8 (2022)
23. Chen, Y., Shi, Y., Zhang, B.: Data-driven optimal voltage regulation using input convex neural networks. Electric Power Systems Research (2020)
24. Cheng, C., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: Automated Technology for Verification and Analysis (ATVA) (2017)
25. Conforti, M., Cornuéjols, G., Zambelli, G.: Integer programming. Springer (2014)
26. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems (1989)
27. Delarue, A., Anderson, R., Tjandraatmadja, C.: Reinforcement learning with combinatorial actions: An application to vehicle routing. In: NeurIPS (2020)
28. Demirović, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., Stuckey, P.J.: MurTree: Optimal decision trees via dynamic programming and search. Journal of Machine Learning Research (2022)
29. Dumouchelle, J., Julien, E., Kurtz, J., Khalil, E.B.: Neur2RO: Neural two-stage robust optimization. arXiv:2310.04345 (2023)
30. Dumouchelle, J., Patel, R., Khalil, E.B., Bodur, M.: Neur2SP: Neural two-stage stochastic programming. Neural Information Processing Systems (NeurIPS) (2022)
31. ElAraby, M., Wolf, G., Carvalho, M.: OAMIP: Optimizing ANN architectures using mixed-integer programming. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR) (2023)
32. Fajemisin, A., Maragno, D., den Hertog, D.: Optimization with constraint learning: A framework and survey. European Journal of Operational Research (2023)
33. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints (2018)
34. Florio, A.M., Martins, P., Schiffer, M., Serra, T., Vidal, T.: Optimal decision diagrams for classification. In: AAAI Conference on Artificial Intelligence (AAAI) (2023)
35. Funahashi, K.I.: On the approximate realization of continuous mappings by neural networks. Neural Networks (1989)
36. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: International Conference on Artificial Intelligence and Statistics (AISTATS) (2011)
37. Günlük, O., Kalagnanam, J., Li, M., Menickelly, M., Scheinberg, K.: Optimal decision trees for categorical data via integer programming. Journal of Global Optimization (2021)
38. Gurobi: Gurobi Machine Learning. https://github.com/Gurobi/gurobi-machinelearning (2023), accessed: 2023-12-03
39. Hahnloser, R., Sarpeshkar, R., Mahowald, M., Douglas, R., Seung, S.: Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature (2000)
40. Hanin, B., Rolnick, D.: Complexity of linear regions in deep networks. In: International Conference on Machine Learning (ICML) (2019)

41. Hanin, B., Rolnick, D.: Deep ReLU networks have surprisingly few activation patterns. In: Neural Information Processing Systems (NeurIPS). vol. 32 (2019)
42. Hanin, B., Sellke, M.: Approximating continuous functions by ReLU nets of minimal width. arXiv:1710.11278 (2017)
43. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks (1989)
44. Hu, H., Huguet, M.J., Siala, M.: Optimizing binary decision diagrams with maxsat for classification. In: AAAI Conference on Artificial Intelligence (AAAI) (2022)
45. Hu, X., Rudin, C., Seltzer, M.: Optimal sparse decision trees. Neural Information Processing Systems (NeurIPS) (2019)
46. Huchette, J., Muñoz, G., Serra, T., Tsay, C.: When deep learning meets polyhedral theory: A survey. arXiv:2305.00241 (2023)
47. Icarte, R.T., Illanes, L., Castro, M.P., Ciré, A.A., McIlraith, S.A., Beck, J.C.: Training binarized neural networks using MIP and CP. In: International Conference on Principles and Practice of Constraint Programming (CP) (2019)
48. Kanamori, K., Takagi, T., Kobayashi, K., Ike, Y., Uemura, K., Arimura, H.: Ordered counterfactual explanation by mixed-integer linear optimization. In: AAAI Conference on Artificial Intelligence (AAAI) (2021)
49. Kronqvist, J., Li, B., Rolfes, J., Zhao, S.: Alternating mixed-integer programming and neural network training for approximating stochastic two-stage problems. arXiv:2305.06785 (2023)
50. Kurtz, J., Bah, B.: Efficient and robust mixed-integer optimization methods for training binarized deep neural networks. arXiv:2110.11382 (2021)
51. Lawless, C., Dash, S., Günlük, O., Wei, D.: Interpretable and fair boolean rule sets via column generation. Journal of Machine Learning Research **24**(229), 1–50 (2023)
52. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998)
53. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature (2015)
54. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J., et al.: Algorithms for verifying deep neural networks. Foundations and Trends® in Optimization **4**(3-4), 244–404 (2021)
55. Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L.: The expressive power of neural networks: A view from the width. In: Neural Information Processing Systems (NeurIPS) (2017)
56. Maragno, D., Wiberg, H., Bertsimas, D., Birbil, S.I., Hertog, D.d., Fajemisin, A.: Mixed-integer optimization with constraint learning. Operations Research (2023)
57. McDonald, T., Tsay, C., Schweidtmann, A.M., Yorke-Smith, N.: Mixed-integer optimisation of graph neural networks for computer-aided molecular design. arXiv:2312.01228 (2023)
58. Montúfar, G.: Notes on the number of linear regions of deep neural networks. In: Sampling Theory and Applications (SampTA) (2017)
59. Montúfar, G., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. In: Neural Information Processing Systems (NeurIPS). vol. 27 (2014)
60. Murzakhanov, I., Venzke, A., Misyris, G.S., Chatzivasileiadis, S.: Neural networks for encoding dynamic security-constrained optimal power flow. In: Bulk Power Systems Dynamics and Control Sympositum (2022)
61. Nair, V., Hinton, G.: Rectified linear units improve restricted boltzmann machines. In: International Conference on Machine Learning (ICML) (2010)

62. Park, S., Yun, C., Lee, J., Shin, J.: Minimum width for universal approximation. In: International Conference on Learning Representations (ICLR) (2021)
63. Pascanu, R., Montúfar, G., Bengio, Y.: On the number of response regions of deep feedforward networks with piecewise linear activations. In: International Conference on Learning Representations (ICLR) (2014)
64. Patil, V., Mintz, Y.: A mixed-integer programming approach to training dense neural networks. arXiv:2201.00723 (2022)
65. Perakis, G., Tsiourvas, A.: Optimizing objective functions from trained relu neural networks via sampling. arXiv:2205.14189 (2022)
66. Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., Dickstein, J.: On the expressive power of deep neural networks. In: International Conference on Machine Learning (ICML) (2017)
67. Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions. In: ICLR Workshop Track (2018)
68. Rosenhahn, B.: Mixed integer linear programming for optimizing a Hopfield network. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD) (2022)
69. Rössig, A., Petkovic, M.: Advances in verification of ReLU neural networks. Journal of Global Optimization **81**, 109–152 (2021)
70. Say, B., Wu, G., Zhou, Y.Q., Sanner, S.: Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In: International Joint Conference on Artificial Intelligence (IJCAI) (2017)
71. Serra, T., Ramalingam, S.: Empirical bounds on linear regions of deep rectifier networks. In: AAAI Conference on Artificial Intelligence (AAAI) (2020)
72. Serra, T., Tjandraatmadja, C., Ramalingam, S.: Bounding and counting linear regions of deep neural networks. In: International Conference on Machine Learning (ICML) (2018)
73. Serra, T., Yu, X., Kumar, A., Ramalingam, S.: Scaling up exact neural network compression by ReLU stability. In: Neural Information Processing Systems (NeurIPS) (2021)
74. Serra, T., Kumar, A., Ramalingam, S.: Lossless compression of deep neural networks. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR) (2020)
75. Strong, C.A., Wu, H., Zeljić, A., Julian, K.D., Katz, G., Barrett, C., Kochenderfer, M.J.: Global optimization of objective functions represented by ReLU networks. Machine Learning (2021)
76. Telgarsky, M.: Representation benefits of deep feedforward networks. arXiv:1509.08101 (2015)
77. Thorbjarnarson, T., Yorke-Smith, N.: Optimal training of integer-valued neural networks with mixed integer programming. PLoS ONE (2023)
78. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: International Conference on Learning Representations (ICLR) (2019)
79. Tong, J., Cai, J., Serra, T.: Relax-and-Walk Implementation. https://github.com/JiataiTong/Optimization-Over-Trained-Neural-Networks-Taking-a-Relaxing-Walk (2024), accessed: 2024-01-28
80. Tsay, C., Kronqvist, J., Thebelt, A., Misener, R.: Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. In: Neural Information Processing Systems (NeurIPS). vol. 34 (2021)
81. Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C.G., Schaus, P.: Learning optimal decision trees using constraint programming. Constraints (2020)

82. Verwer, S., Zhang, Y.: Learning decision trees with flexible constraints and objectives using integer optimization. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR) (2017)
83. Verwer, S., Zhang, Y.: Learning optimal classification trees using a binary linear program formulation. In: AAAI Conference on Artificial Intelligence (AAAI) (2019)
84. Wang, Y.: Estimation and comparison of linear regions for relu networks. In: International Joint Conference on Artificial Intelligence (IJCAI) (2022)
85. Wu, G., Say, B., Sanner, S.: Scalable planning with deep neural network learned transition models. Journal of Artificial Intelligence Research (2020)
86. 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 (ICLR) (2019)
87. Yang, S., Bequette, B.W.: Optimization-based control using input convex neural networks. Computers & Chemical Engineering (2021)
88. Yarotsky, D.: Error bounds for approximations with deep ReLU networks. Neural Networks (2017)
89. Zhu, H., Murali, P., Phan, D., Nguyen, L., Kalagnanam, J.: A scalable MIP-based method for learning optimal multivariate decision trees. Neural Information Processing Systems (NeurIPS) (2020)