Optimization over Trained (and Sparse) Neural Networks: A Surrogate within a Surrogate

Hung Pham[1], Aiden Ren[1], Ibrahim Tahir[1], Jiatai Tong[2], and Thiago Serra[3]

[1] Bucknell University, Lewisburg PA, United States
{hqp001,zr002,it005}@bucknell.edu
[2] Northwestern University, Evanston IL, United States
jiataitong2026@u.northwestern.edu
[3] University of Iowa, Iowa City IA, United States
thiago-serra@uiowa.edu

Abstract. In constraint learning, we use a neural network as a *surrogate* for part of the constraints or of the objective function of an optimization model. However, the tractability of the resulting model is heavily influenced by the size of the neural network used as a surrogate. One way to obtain a more tractable surrogate is by pruning the neural network first. In this work, we consider how to approach the setting in which the neural network is actually a given: how can we solve an optimization model embedding a large and predetermined neural network? We propose surrogating the neural network itself by pruning it, which leads to a sparse and more tractable optimization model, for which we hope to still obtain good solutions with respect to the original neural network. For network verification and function maximization models, that indeed leads to better solutions within a time limit, especially—and surprisingly—if we skip the standard retraining step known as finetuning. Hence, a pruned network with worse inference for lack of finetuning can be a better surrogate.

Keywords: Constraint learning · Neural network pruning · Neural network verification · Piecewise linear approximation · Rectified linear units.

1 Introduction

In the last five years, we have seen a growing interest in approximating a constraint or an objective function of an optimization model with a neural network. This approach is often denoted as *constraint learning*. The most compelling circumstance for using constraint learning is when the exact form of some constraints or part of the objective function is unknown, but can be approximated using available data. When the exact form is known, another compelling circumstance is when the formulation is intractable for a combination of factors such as being nonlinear, nonconvex, and very large [87, 112, 131, 144].

Constraint learning has been applied in optimization models for scholarship allocation [8], chemotherapy [83], molecular design [85], power grid operation [18, 76, 92], and automated control in general [106, 114, 135, 139]. We can also use optimization models involving neural networks to evaluate those neural networks

for adversarial perturbations [2,19,23,104,116], compression [30,110,111], counterfactual explanations [66, 125], equivalence [67], expressiveness [16, 108, 109], monotonicity [75], and reachability [28,77]. Conversely, when the neural network is trained as a reinforcement learning policy, the constraints in the optimization model can be used for limiting the action space [13, 24]. Many frameworks have been proposed to embed neural networks and other machine learning models as part of optimization models, such as JANOS [8], reluMIP [78], OMLT [17], OCL [31], OptiCL [83], Gurobi Machine Learning [42], and PySCIPOpt-ML [127].

Some neural network architectures are more convenient to embed in optimization models than others. In particular, we typically use the Rectified Linear Unit (ReLU) activation function [44, 93] for a couple of reasons. First, this activation function became widely popular in the 2010s due to its good classification performance and relatively lower training cost [38, 70, 93]. Similar to what was already known for other neural network architectures much earlier [22, 36, 60], we now know that ReLU networks are also universal function approximators [47, 140]. More recently, Gaussian Error Linear Units (GELUs) [52] have been used from the very beginning in many foundation models based on the transformer architecture [128], such as GPT [99], BERT [26], and ViT [27]. Nevertheless, a ReLU is a piecewise linear approximation of a GELU, which brings us to the next point. Second, and perhaps most importantly for the continued use of ReLUs nowadays, each neuron with ReLU activation represents a piecewise linear function—and by consequence a neural network with only ReLUs is also a piecewise linear function [3]. In the context of nonlinear optimization, substantial work has been devoted to using piecewise linear approximations in surrogate models [5,21,32,63,81,82,84,86,103,130,133] because we can resort to linear optimization for iterative improvements over such approximations. In fact, there are many active lines of research on what piecewise linear representations can be obtained from different neural network architectures, most of which focused on ReLUs [3,4,16,40,43,45,46,53–55,62,88–90,95,97,100,102,105,108,109,119,124,132].

With a piecewise linear representation, we can embed the neural network in the optimization model using a Mixed-Integer Linear Programming (MILP) formulation. However, such formulations range from being small but having a weak linear relaxation to having a stronger relaxation but being prohibitively large [61], which makes it difficult to use an off-the-self MILP solver. That motivated studies on how to tackle such models, ranging from reformulation [61, 76, 107, 123] to methods for obtaining smaller big M coefficients through activation bounds [6, 19, 33, 41, 57, 74, 114, 123, 143] and parameter rescaling [98], activation inferences for fixing variables and limiting search space [11, 108, 120, 138], cutting planes [2], and heuristic solutions [96, 121]. Another—perhaps overlooked—approach is using neural networks that are smaller [15] and sparser [106, 138].

In this work, we explore this latter approach of using sparser neural networks. On the one hand, we know that (i) sparser optimization models tend to be solved faster; and (ii) we can sparsify a neural network, which is known as *network pruning*, and still recover its performance by retraining for a few steps, which is known as *finetuning*. In fact, it is already reasonably well established

that pruning a neural network in advance leads to a more tractable constraint learning model [106,138]. On the other hand, finetuning requires access to training data and requires a non-negligible runtime overhead. Moreover, if the purpose of solving an optimization model that embeds a neural network is to verify properties of the neural network itself, then it is not immediate if—and how—to use a pruned counterpart of the given neural network as a proxy. Therefore, we evaluate how much we can prune, and how much we should finetune, a neural network to still obtain effective and efficient surrogate optimization models.

2 From Notation to Formulation

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, for which reason we abstract that possibility. We also abstract that fully-connected layers are often followed by a softmax layer [12], since the largest output of softmax matches the largest input of softmax, for which reason it is not necessary to include the softmax layer in applications such as network verification.

Each 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 $h_i^l = \max\{0, g_i^l\}$. When training the neural network, we vary the paramenters in $\{(\boldsymbol{W}^l, \boldsymbol{b}^l)\}_{l \in \mathbb{L}}$ to better fit the values given for the set of input–output pairs $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i \in \mathbb{S}}$ representing the training set.

When optimizing over the trained neural network, we flip what is variable and what is constant in relation to training: we vary the input $\boldsymbol{x} = \boldsymbol{h}^0$ and the outputs at each layer $\{(\boldsymbol{g}^l, \boldsymbol{h}^l)\}_{l \in \mathbb{L}}$ for the fixed parameters $\{(\boldsymbol{W}^l, \boldsymbol{b}^l)\}_{l \in \mathbb{L}}$. For each layer $l \in \mathbb{L}$ and neuron $i \in \mathbb{N}_l$, we also use a binary variable $\boldsymbol{z}_i^l$ representing the ReLU activation to map inputs to outputs using MILP:

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

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

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

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

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

The indicator constraints (2)–(3) can be modeled with big M constraints [10].

In the absence of other decision variables, one general form of representing a linear optimization model embedding a neural network is as follows:

$$\max \quad \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{d}^T\boldsymbol{y} \tag{6a}$$
$$\text{s.t.} \quad A\boldsymbol{x} + B\boldsymbol{y} \leq b \tag{6b}$$
$$\boldsymbol{y} = \text{NN}(\boldsymbol{x}) \tag{6c}$$

We use $\boldsymbol{y} = \text{NN}(\boldsymbol{x})$ as a shorthand for the input–output mapping defined by the set of constraints (1)–(5) $\forall l \in \mathbb{L}, i \in \mathbb{N}_l$ across the entire neural network.

Next, we describe two applications based on special cases of this formulation. They are both used for evaluating our approach.

2.1 Network Verification

For neural networks used for classification, one application of embedding them in optimization models is to determine if there is an adversarial perturbation for a chosen input $\boldsymbol{x}^{(i)}, i \in \mathbb{S}$ from the training set. If $\boldsymbol{x}^{(i)}$ is correctly classified as class $j \in \{1, \ldots, m\}$ by having an output $\boldsymbol{y}$ such that $y_j^{(i)} > y_k^{(i)}$ $\forall k \in \{1, \ldots, m\} \setminus \{j\}$, then we can try to determine if there is a similar input $\boldsymbol{x}$ with a different classification. By varying the inputs within $\{\boldsymbol{x} : \|\boldsymbol{x} - \boldsymbol{x}^{(i)}\|_1 \leq \varepsilon\}$ for a chosen ε, we try to find an input that is better classified with a chosen class $j' \neq j$, i.e., $y_{j'} > y_j$.[4] That leads to the following MILP formulation:

$$\max \quad y_{j'} - y_j \tag{7a}$$
$$\text{s.t.} \quad \sum_{k \in \{1,\ldots,n_0\}} |\boldsymbol{x}_k - \boldsymbol{x}_k^{(i)}| \leq \varepsilon \tag{7b}$$
$$\boldsymbol{y} = \text{NN}(\boldsymbol{x}) \tag{7c}$$

In the model above, any solution with a positive objective function value entails an adversarial perturbation; whereas a nonpositive optimal value implies that no such perturbation exists for the given choices of i, j', and ε.

2.2 Function Maximization

For neural networks used for regression, one application of embedding them in optimization models is to optimize over the function surrogated by the neural network. The case of finding the maximum for a neural network with a single output—i.e., $m = 1$—leads to the following MILP formulation:

$$\max \quad y_1 \tag{8a}$$
$$\text{s.t.} \quad \boldsymbol{y} = \text{NN}(\boldsymbol{x}) \tag{8b}$$
$$\boldsymbol{x} \in \mathbb{X} \tag{8c}$$

[4] With j' fixed, we are only ensuring that j' would be a better classification than j, since there might be another class j'' dominating both, i.e., $y_{j''} > y_{j'} > y_j$.

3 From Network Pruning to Sparse Surrogates

Neural networks have a very peculiar trait: assuming that two neural networks can be trained to achieve the same level of accuracy, it is often easier to train the largest one than it is to train the smallest one. But after training a neural network that is larger than it needs to be, we can simplify the network by removing neurons or connections and then still recover a similar accuracy by carefully adjusting the remaining parameters. In fact, this is becoming mainstream knowledge with the constant discussion about number of parameters in large language models and the application of pruning techniques for obtaining comparable variants that are smaller and faster for inference [35, 80, 117, 136].

3.1 Background

Neural networks are pruned by either (i) removing connections, which is equivalent to zeroing out specific parameters—known as *unstructured* pruning; or (ii) removing units, such as neurons, convolutional filters, or layers—known as *structured* pruning. The latter has greater appeal for performance, since it goes beyond reducing storage to using smaller hardware and running the model faster, but then we need to prune less to still recover the original performance [20].

But why do we need network pruning? A larger neural network has a smoother loss landscape [72,118], which facilitates training convergence; and the larger size may also prevent layers from becoming inactive [101], which is a common cause for unsuccessful training. **Why does network pruning work?** In larger networks, there is redundancy among the parameters [25], and zeroing out parameters leads to a loss landscape from which finetuning the pruned network for recovering the original performance converges considerably faster than the original training [65]. From a model flexibility perspective, unstructured pruning at moderate rates has little effect on the expressiveness of the neural network architecture [16]. **How much can we prune?** In sufficiently large networks, we can remove as much as half of the parameters and still recover the original performance—or even improve upon it [56]. However, that varies with the task for which the network is trained [73]. Moreover, pruning may have a disparate effect across classes [58, 59, 94], which may lead to pruned networks that exacerbate existing performance differences [122]. On the bright side, a smaller amount of pruning may actually correct such distortions [39]. **What should we prune?** The two main philosophies [9] are (1) to remove parameters with the smallest absolute value—dating back to [48, 64, 91]; and (2) to remove parameters with the smallest expected impact on the output—dating back to [49, 50, 69], and including the special case of exact compression [37, 110, 111, 115]. **And when should we prune?** Most studies have focused on pruning once (*one-shot*) and after training, but recent work has shown that it might be beneficial to prune iteratively and during training [34], or even before training [71].

Mathematical optimization has been extensively used in more sophisticated pruning methods [1, 7, 14, 16, 29, 30, 49–51, 69, 79, 110, 111, 113, 129, 134, 141, 142]. Those methods tend to perform better than simpler heuristics at higher pruning

rates. However, they also come at a greater computational cost. For moderate pruning, something as simple as removing the weights with the smallest absolute values, known as *Magnitude Pruning* (MP) [48, 64, 91], remains a very competitive choice [142]. We take the liberty of calling it "unreasonably effective".

The use of pruned neural networks in mathematical optimization, however, has been less explored. Among the first studies on embedding neural networks, Say et al. [106] observed that a modest amount of unstructured pruning—removing about 20% of the parameters—significantly reduced the runtime for solving the optimization model. Moreover, Xiao et al. [138] and Cacciola et al. [15] observed that structured pruning—having fewer neurons and therefore fewer binary decision variables mapping the activation state of each neuron—leads to comparable neural networks that are more easily verified.

3.2 The Sparse Surrogate Approach

Suppose that we have a (dense) neural network $\mathcal{D}$ to which $\mathcal{S}$ is a sparse counterpart obtained by network pruning, with $\boldsymbol{y}^D = \mathcal{D}(\boldsymbol{x})$ and $\boldsymbol{y}^S = \mathcal{S}(\boldsymbol{x})$ as the corresponding outputs from those two neural networks for a same input $\boldsymbol{x}$.

We will succinctly describe how to use $\mathcal{S}$ for obtaining solutions for constraint learning models on $\mathcal{S}$. We will use the models from Sections 2.1 and 2.2.

Network Verification First, we consider the case of a network verification problem, in which we validate if an adversarial input to the pruned network is an adversarial input to the original network. Let $\text{VNN}(\mathcal{N}, \boldsymbol{x}^{(i)}, \varepsilon, j, j')$ be the MILP formulation (7a)–(7c) for verifying neural network $\mathcal{N} \in \{\mathcal{D}, \mathcal{S}\}$ starting from input $\boldsymbol{x}^{(i)}$ and with maximum $L1$-norm distance ε for obtaining another input $\boldsymbol{x}$ in which the output for class j' is as large as possible in comparison to that of class j, i.e., we want to find an input $\boldsymbol{x}$ maximizing $y^N_{j'} - y^N_j$ for $\boldsymbol{y}^N = \mathcal{N}(\boldsymbol{x})$. For the purpose of verification, it suffices to find an $\boldsymbol{x}$ such that $y^N_{j'} > y^N_j$. To find a solution with positive value for dense model $\mathbf{VNN}(\mathcal{D}, \varepsilon, x^{(i)}, j, j')$, we resort to solving sparse model $\mathbf{VNN}(\mathcal{S}, \varepsilon, x^{(i)}, j, j')$ as outlined in Algorithm 1.

Algorithm 1 Heuristic for obtaining an adversarial input to a (dense) neural network $\mathcal{D}$ by trying to solve the same problem on its sparse counterpart $\mathcal{S}$

```
while trying to solve VNN(S, ε, x^(i), j, j') do        ▷ MILP solver call
    if solution (x, y^S) found then                      ▷ Feasible solution callback
        y^D = D(x)                                       ▷ Output of the dense model D
        if y^D_{j'} > y^D_j then                         ▷ Check if x is adversarial to D
            return x                                     ▷ Adversarial input found
        end if
    end if
end while
return ∅                                                 ▷ No adversarial input found
```

Function Maximization Next, we consider the case of a function maximization problem, in which we check if an input to the pruned network yields a better value than the inputs previously tried on the dense network. Let $\mathbf{FM}(\mathcal{N})$ be the MILP formulation (8a)–(8c) for maximizing the output of the function modeled by network $\mathcal{N} \in \{\mathcal{D}, \mathcal{S}\}$ over domain $\mathbb{X}$. Unlike the network verification case, there is no criterion for prematurely stopping the search in this application. To find a solution with better value for dense model $\mathbf{FM}(\mathcal{D})$, we resort to solving sparse model $\mathbf{FM}(\mathcal{S})$ as outlined in Algorithm 2.

Algorithm 2 Heuristic for obtaining an adversarial input to a (dense) neural network $\mathcal{D}$ by trying to solve the same problem on its sparse counterpart $\mathcal{S}$

$\boldsymbol{x}^* \leftarrow \emptyset$ ▷ Initialize best solution as none
$y^* \leftarrow -\infty$ ▷ Placeholder for best solution value
while trying to solve $\mathbf{FM}(\mathcal{S})$ **do** ▷ MILP solver call
  if solution $(\boldsymbol{x}, y^S)$ found **then** ▷ Feasible solution callback
    $y^D = \mathcal{D}(\boldsymbol{x})$ ▷ Output of the dense model $\mathcal{D}$
    if $\boldsymbol{x}^* = \emptyset$ or $y^D > y^*$ **then** ▷ Check if $\boldsymbol{x}$ is the first or a better solution
      $\boldsymbol{x}^* \leftarrow \boldsymbol{x}$ ▷ Update best solution
      $y^* \leftarrow y^D$ ▷ Update best solution value
    end if
  end if
end while
return $\boldsymbol{x}^*$ ▷ Provide best solution found

4 Experiments for Network Verification

We evaluated the time for finding an adversarial input for a (dense) neural network $\mathcal{D}$ by directly solving model $\text{VNN}(\mathcal{D}, \boldsymbol{x}^{(i)}, \varepsilon, j, j')$, which we denote as *Dense Runtime*, in comparison to indirectly solving model $\text{VNN}(\mathcal{S}, \boldsymbol{x}^{(i)}, \varepsilon, j, j')$ while resorting to Algorithm 1, which we denote as *Pruned Runtime*. Our goal is to find if, and when, Pruned Runtime is shorter than Dense Runtime.

4.1 Technical Details

We used the source code of SurrogateLIB [126] as the basis for training neural networks and producing network verification problems, starting with the MNIST dataset [68] and then extending the code to also work with the Fashion-MNIST dataset [137]. For each of those datasets, we tried all combined variations of inputs sizes $n_0 = 18^2$ (compressed images) and $n_0 = 28^2$ (images at original size), number of ReLU layers $L \in \{2, 4\}$, and uniform layer width $n_i \in \{32, 64\}$ $\forall i \in \{1, \ldots, L\}$. For each dataset and choice of hyperparameters, we used 10 randomization seeds for training neural networks, associating

them with verification problems on distinct samples from the training set, and choosing some $\varepsilon \in [4.5, 5.5]$. In total, we have 160 verification problems.

For producing pruned versions of those networks, we used the PyTorch library. On the number of parameters removed, we applied layerwise pruning rates of 0.3, 0.5, 0.8, 0.9, and 0.95. On what to prune, we applied both Magnitude Pruning (MP), which corresponds to pruning the parameters with the smallest absolute value; and Random Pruning (RP), which corresponds to pruning parameters randomly. In addition, we also evaluated *unstructured pruning*, in which case the parameters across the layer are pruned indiscriminately; and *structured pruning*, in which case we consider all the parameters associated with a neuron and decide about pruning whole neurons instead. Finally, as a last step we also opted between finetuning the pruned network or not finetuning it and keeping it as it was after being pruned. For each of the 160 original verification problems, the combination of all the pruning choices above resulted in solving variants in 40 pruned versions of each neural network. In total, we use 6,560 models.

We used the BisonNet cluster. The steps involving the solution of MILP models with Gurobi were run on Intel Xeon Gold 6442Y CPUs. The steps involving network training and pruning were run on AMD EPYC 7252 CPUs with NVIDIA RTX A5000 GPUs. The neural networks were trained and pruned using Torch 2.0.0. Each model on MNIST was trained for 5 epochs, and each model on Fashion-MNIST was trained for 40 epochs. Without finetuning, there was a single round of pruning. With finetuning, there were 5 pruning rounds and each round had 5 epochs of retraining. The MILP models were solved using Gurobi Optimizer 10.0.1 with a time limit of 300 seconds for each of the 6,560 models.

4.2 Results and Analysis

Our best results were obtained by using Algorithm 1 with unstructured MP instead of solving the verification model directly. We compare the runtimes (in seconds) to solve the verification model directly and indirectly for MNIST in Figure 1 and Fashion-MNIST in Figure 2. We vary pruning rate, whether finetuning is used, and whether finetuning time is counted as part of the runtime. We report the percentage of instances above and below the identity line, corresponding to the direct and the indirect approaches being faster. Those percentages do not add up to 100% if there are ties, which includes when both methods time out.

We draw the following conclusions and provide a rationale to each conclusion by listing observations about the plots in those figures:

(I) Using our approach in network verification is advantageous in terms of (i) individual runtimes as well as (ii) number of instances solved:
  (i) We found adversarial inputs faster for most instances regardless of pruning rate and of finetuning the pruned neural networks or not.
  (ii) When there are timeouts (i.e., not finding an adversarial input) from solving the verification problem directly (as with MNIST), then our approach with a small pruning rate reduces the number of timeouts.

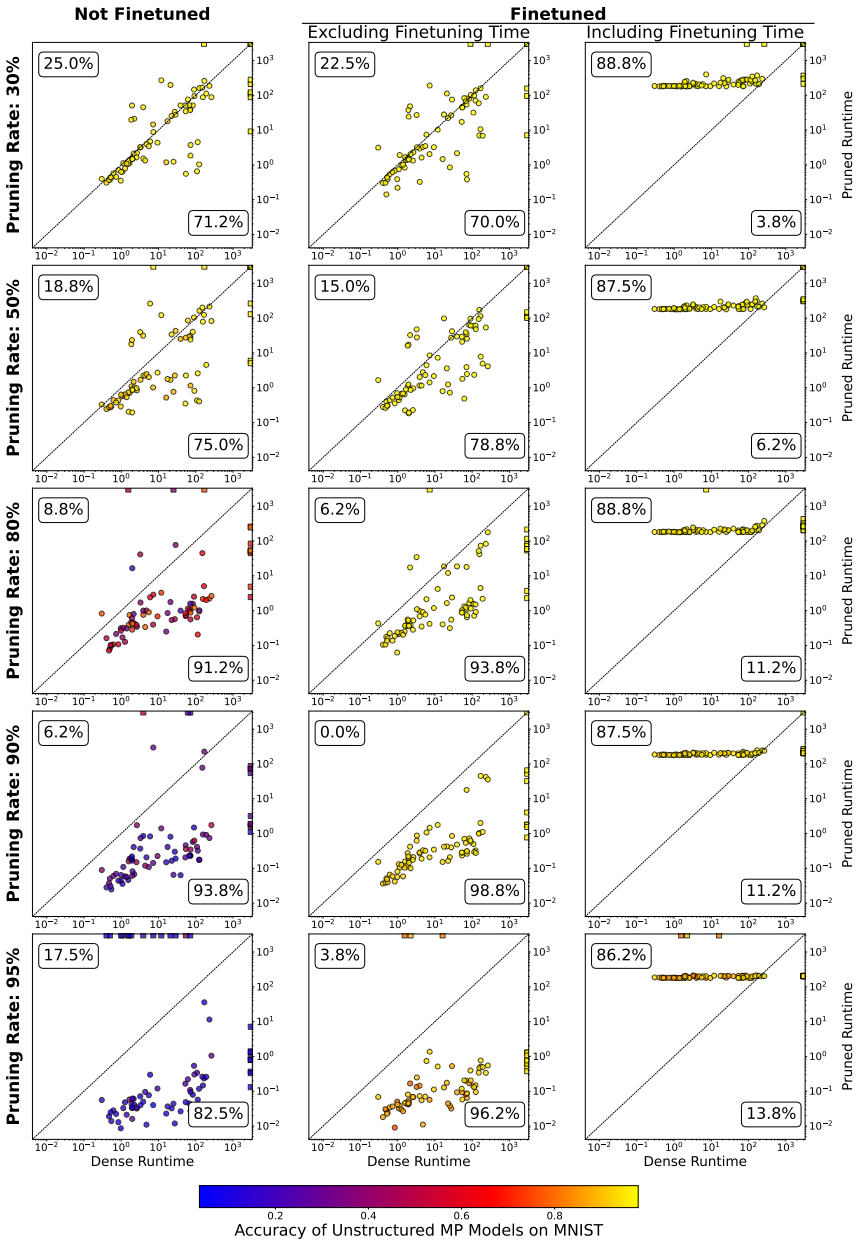

Fig. 1. Time to find adversarial input to networks trained on MNIST by solving the verification problem directly (x axis) or indirectly with Algorithm 1 (y axis) per pruning rate, use of finetuning, and inclusion of finetuning in runtime. Squares on top or (and) right sides indicate no adversarial input found for either (both). Ties are not counted.

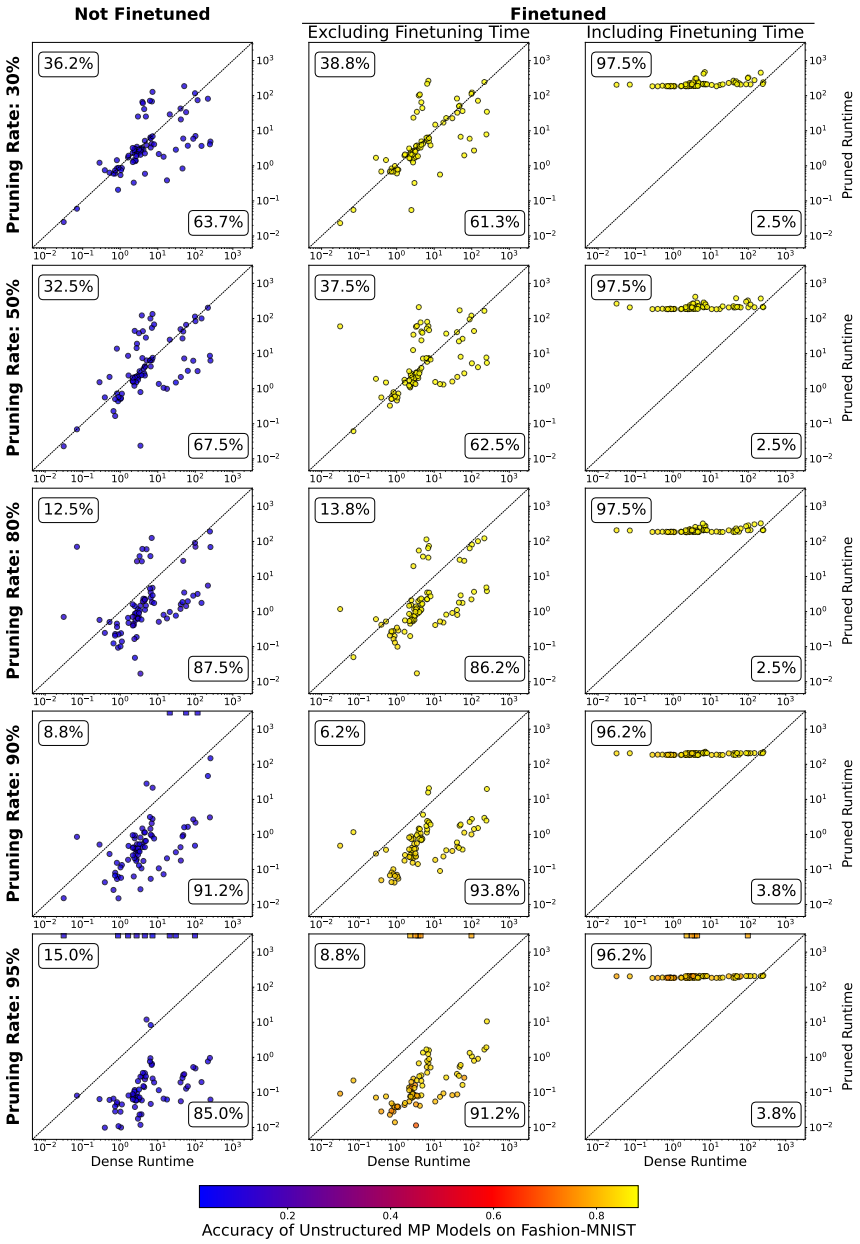

Fig. 2. Time to find adversarial input to networks on Fashion-MNIST by solving the verification problem directly (x axis) or indirectly with Algorithm 1 (y axis) per pruning rate, use of finetuning, and inclusion of finetuning in runtime. Squares on top or (and) right sides indicate no adversarial input found for either (both). Ties are not counted.

(II) The pruning rate can be adjusted for different purposes, from finding more adversarial inputs up to a time limit at lower rates (ii again) to finding (iii) most adversarial inputs faster at higher rates and (iv) fewer adversarial inputs but in a much shorter amount of time at the highest rate:
  (iii) The number of runtime improvements increases with pruning rate up to 90% for both datasets, finetuned or not, and then decreases.
  (iv) The differences between runtimes become more extreme in our favor with greater pruning rates, but then the timeouts also increase.

(III) The pruned neural network does not need to be a good classifier to help us find adversarial inputs (v). In fact, it is not helpful to finetune the network to improve accuracy after pruning at lower pruning rates (vi). For higher pruning rates, finetuning the neural network can be helpful (vii), but the cost of finetuning would have to be amortized over solving multiple verification problems on the same neural network (viii):
  (v) The lower accuracy in the case without finetuning, almost approaching random guessing (10% on either dataset), did not prevent us from using the pruned neural networks for obtaining adversarial inputs.
  (vi) The results were better without finetuning for the lowest pruning rates (one for MNIST and three for Fashion-MNIST).
  (vii) The percentage difference of instances solved faster with finetuning is only on the second most significant digit for the higher pruning rate, except the highest (e.g., 98.8% instead of 93.5% on MNIST and 93.8% instead of 91.2% on Fashion-MNIST for pruning rate 90%).
  (viii) If we account for the finetuning cost, then it is generally faster to solve the verification problem directly.

Given the cost–benefit advantage of not using finetuning, we conducted the following ablations restricted to the results without finetuning.

First, we considered the impact of other network pruning choices. Figure 3 compares the joint results on MNIST and Fashion-MNIST by using unstructured MP as before, then by only replacing unstructured with structured pruning, and then by only replacing MP with RP. Considering possible questions due to the favorable results for structured pruning in some of the plots, Table 1 summarizes the number of instances on both datasets in which an adversarial input would

Table 1. Percentage of instances for which solving the verification problem indirectly on a pruned network is faster by pruning rate (columns); using unstructured and structured pruning (top and bottom rows); and not finetuning (NF) or finetuning (F). Unfavorable figures (below 50%) are reported in italics for greater emphasis.

		Pruning Rate				
		0.3	0.5	0.8	0.9	0.95
Unstructured	NF	67.5%	71.3%	89.4%	92.5%	83.8%
	F	65.6%	70.6%	90.0%	96.3%	93.8%
Structured	NF	75.0%	78.8%	70.6%	72.5%	65.0%
	F	59.4%	57.5%	56.3%	*49.4%*	*46.3%*

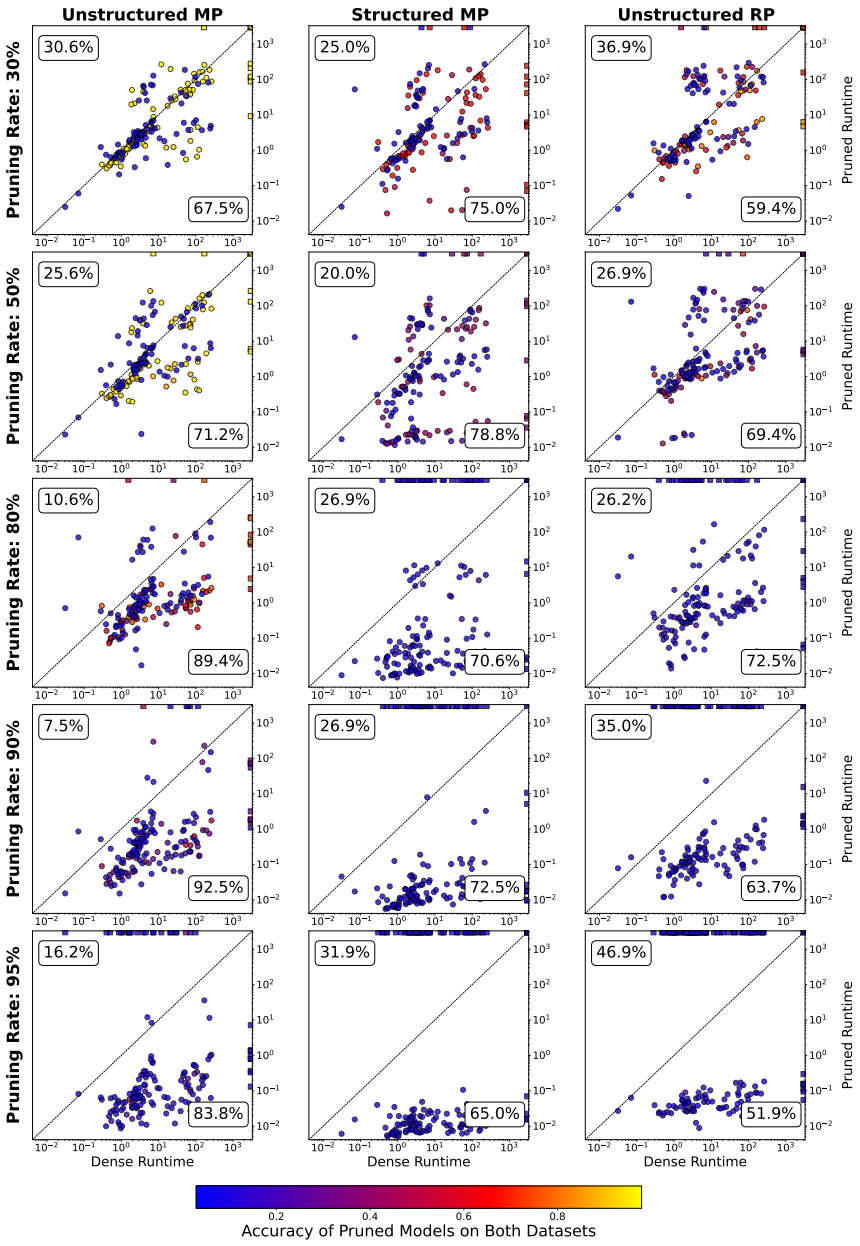

Fig. 3. Time to find adversarial input to networks on MNIST and Fashion-MNIST by solving the verification problem directly (x axis) or indirectly with Algorithm 1 (y axis) per pruning rate for different forms of network pruning. Squares on top or (and) right sides indicate no adversarial input found for either (both). Ties are not counted.

Table 2. Percentage of instances for which solving the verification problem indirectly on a pruned network is faster by pruning rate (columns); disaggregated in terms of input size (top rows), number of layers (middle), and neurons per layer (bottom rows).

		Pruning Rate				
		0.3	0.5	0.8	0.9	0.95
Input Size	18^2	75.0%	80.6%	86.9%	84.4%	76.3%
	28^2	67.5%	69.4%	73.1%	80.6%	72.5%
Network Depth	2	80.6%	84.4%	86.3%	86.3%	76.3%
	4	61.9%	65.6%	73.8%	78.8%	72.5%
Layer Width	32	69.4%	78.8%	78.1%	81.9%	75.6%
	64	73.1%	71.3%	81.9%	83.1%	73.1%

have been found faster than by directly solving the verification problem according to the structural type of pruning and the use of finetuning.

The purpose of considering RP, which is not an appealing choice intuitively, was to evaluate if a deliberate choice on how to prune would influence the results. We can see that it does—and that the only deliberate choice that we considered was MP. We opted for MP due to its small computational cost and widely regarded effectiveness in practice. Hence, we leave is open to future work if it would be beneficial to replace MP with a more sophisticated network pruning algorithm. That improvement seems reasonable to expect, but we believe that it would be beyond the scope of this particular study.

Second, we considered how the results vary based on the dimensions of the neural networks considered. Table 2 summarizes the number of instances on both datasets in which an adversarial input would have been found faster than by directly solving the verification problem according to input size, number of layers (network depth), and neurons per layer (layer width).

We draw more conclusions and justify them based on the above ablations:

(IV) Structured pruning without finetuning is also potentially applicable at lower pruning rates (ix), with the caveat that more timeouts may occur (x). However, finetuning after structured pruning appears to make the pruned networks significantly different from the original neural network, since their adversarial inputs are less compatible (xi):
  (ix) For the two lowest pruning rates (up to 50%), more instances are solved faster with structured MP than with unstructured MP.
  (x) Across all pruning rates, structured pruning leads to more timeouts.
  (xi) Finetuning has a positive effect when applied to neural networks that were subject to unstructured pruning, and that positive effect grows with pruning rate. The opposite happens with structured pruning.

(V) The criteria of what connections to prune (such as MP vs. RP) has a significant effect on the results (xii), and may help extending this approach to neural networks with larger inputs and more layers (xiii):
  (xii) The results for unstructured MP are better than those for unstructured RP, and the difference grows with the pruning rate.

(xiii) The benefit of our approach is more significant in neural networks with smaller input size or smaller depth. On the other hand, layer width does not affect results in a clearly monotonic way.

5 Experiments for Function Maximization

We evaluated the best solution obtained for a (dense) neural network $\mathcal{D}$ by directly solving model $\mathbf{FM}(\mathcal{D})$, which we denote as *Dense Maximum Value*, in comparison to indirectly solving model $\mathbf{FM}(\mathcal{S})$ while resorting to Algorithm 2, which we denote as *Pruned Maximum Value*. Our goal is to find if, and when, Pruned Maximum Value is greater than Dense Maximum Value.

5.1 Technical Details

We used randomly initialized networks with 5 seeds for all combinations of input sizes $n_0 \in \{100, 1000, 10000\}$, number of layers $L \in \{2, 3, 4, 5\}$, uniform layer width $n_i \in \{50, 100, 200\}$ $\forall i \in \{1, \ldots, L\}$, and with an extra one-neuron output layer. That setting is similar to other papers using this problem [96, 121]. Based on the results from Section 4, we used only unstructured MP without finetuning. When using Algorithm 2, we solved FM($\mathcal{S}$) by setting parameters MIPFocus, PoolSearchMode to 1 and PoolSolutions to 1000, ensuring that a greater number of solutions is obtained. We report the baseline of directly solving FM($\mathcal{S}$) with default parameter values, since it gave better results in this case. We set the time limit to 600 seconds for all models. All other settings are as in Section 4.

5.2 Results and Analysis

Our approach based on Algorithm 2 generally produced better results for the instances that would be typically regarded as harder to solve for having a larger number of linear regions [121]. The results are not as favorable as they were for network verification, for which reason we use less space reporting and analyzing them. Nevertheless, they have some interesting similarities and differences when compared to those in Section 4. Table 3 summarizes the number of networks for which we found better values indirectly with Algorithm 2 according to input size, number of layers (network depth), and neurons per layer (layer width).

We draw the following conclusions and justifications based on the results:

(A) The results from Algorithm 2 are generally better if at least one of the dimensions of the neural network has a larger value (a), and consistently more so in the case of layer width (b).
  (a) In 32 out of 35 cases where any dimension is not the smallest, at least half of the instances have a better solution. In contrast, larger input sizes were less favorable to our approach in the case of network verification.
  (b) Unlike input size and network depth, any larger value for network width corresponds to a favorable case for Algorithm 2. In contrast, network width was the least relevant dimension in network verification.

Table 3. Percentage of instances for which solving the function maximization problem indirectly on a pruned network yields a better solution by pruning rate (columns); disaggregated in terms of input size (top rows), number of layers (middle), and neurons per layer (bottom rows). Favorable figures (above 50%) are reported in bold.

		Pruning Rate				
		0.3	0.5	0.8	0.9	0.95
Input Size	100	33.3%	40.0%	48.3%	50.0%	45.0%
	1,000	**63.3%**	48.3%	**63.3%**	50.0%	50.0%
	10,000	**65.0%**	**58.3%**	**56.7%**	**55.0%**	**78.3%**
Network Depth	2	48.9%	40.0%	24.4%	24.4%	24.4%
	3	**57.8%**	44.4%	**66.7%**	**55.6%**	**51.1%**
	4	**62.2%**	**55.6%**	**66.7%**	**64.4%**	**73.3%**
	5	46.7%	**55.6%**	**66.7%**	**62.2%**	**82.2%**
Layer Width	50	**51.7%**	43.3%	35.0%	31.7%	33.3%
	100	**58.3%**	**51.7%**	**65.0%**	**55.0%**	**60.0%**
	200	**51.7%**	**51.7%**	**68.3%**	**68.3%**	**80.0%**

(B) Increasing along a dimension or along the pruning rate while fixing the other does not necessarily lead to better results (c), but very large networks yield considerably better results when using the highest pruning rate (d).

(c) We see at least non-monotonic variation of percentages if we increase any dimension along the same pruning rate, or if we increase the pruning rate along the same dimension. In contrast, we generally observe a monotonic improvement up to 90% pruning rate in network verification.

(d) In all cases in which one of the dimensions is the largest, the best results are obtained for the highest pruning rate, unlike in network verification.

6 Conclusion

With the goal of solving optimization problems embedding a dense neural network, we tackled those problems indirectly through drastically sparsified neural networks serving as surrogates. Those problems become very difficult to solve as the neural networks grow larger in size, and the surrogate is a pruned version of the same neural network. By making the models sparser, we naturally expect to find solutions faster, and in some cases we do not expect or do not need to necessarily find an optimal solution. We believe that this work contributes to understanding how to tackle constraint learning models more effectively.

We have found that a cost-effective approach is applying unstructured pruning while carefully choosing which connections to prune but not finetuning the pruned network afterwards. We obtained consistently strong results in network verification, even if the surrogate network had very low accuracy. We also obtained encouraging, albeit modest, results in function maximization, but under different conditions than we found them for network verification.

Acknowledgments. The authors were supported by the National Science Foundation grant 2104583, including Jiatai Tong while at Bucknell University.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Aghasi, A., Abdi, A., Nguyen, N., Romberg, J.: Net-Trim: Convex pruning of deep neural networks with performance guarantee. In: NeurIPS (2017)
2. Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P.: Strong mixed-integer programming formulations for trained neural networks. Mathematical Programming (2020)
3. Arora, R., Basu, A., Mianjy, P., Mukherjee, A.: Understanding deep neural networks with rectified linear units. In: ICLR (2018)
4. Averkov, G., Hojny, C., Merkert, M.: On the expressiveness of rational ReLU neural networks with bounded depth. In: ICLR (2025)
5. Babayev, D.A.: Piece-wise linear approximation of functions of two variables. Journal of Heuristics (1997)
6. Badilla, F., Goycoolea, M., Muñoz, G., Serra, T.: Computational tradeoffs of optimization-based bound tightening in ReLU networks. arXiv:2312.16699 (2023)
7. Benbaki, R., Chen, W., Meng, X., Hazimeh, H., Ponomareva, N., Zhao, Z., Mazumder, R.: Fast as CHITA: Neural network pruning with combinatorial optimization. In: ICML (2023)
8. Bergman, D., Huang, T., Brooks, P., Lodi, A., Raghunathan, A.U.: JANOS: An integrated predictive and prescriptive modeling framework. INFORMS Journal on Computing (2022)
9. Blalock, D., Ortiz, J., Frankle, J., Guttag, J.: What is the state of neural network pruning? In: MLSys (2020)
10. Bonami, P., Lodi, A., Tramontani, A., Wiese, S.: On mathematical programming with indicator constraints. Mathematical Programming (2015)
11. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of (relu)-based neural networks via dependency analysis. In: AAAI (2020)
12. Bridle, J.S.: Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition (1990)
13. Burtea, R.A., Tsay, C.: Safe deployment of reinforcement learning using deterministic optimization over neural networks. Computer Aided Chemical Engineering (2023)
14. Cacciola, M., Frangioni, A., Li, X., Lodi, A.: Deep neural networks pruning via the structured perspective regularization. SIAM Journal on Mathematics of Data Science (2023)
15. Cacciola, M., Frangioni, A., Lodi, A.: Structured pruning of neural networks for constraints learning. Operations Research Letters (2024)
16. 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: CPAIOR (2023)
17. 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)

18. Chen, Y., Shi, Y., Zhang, B.: Data-driven optimal voltage regulation using input convex neural networks. Electric Power Systems Research (2020)
19. Cheng, C., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: ATVA (2017)
20. Cheng, Y., Wang, D., Zhou, P., Zhang, T.: Model compression and acceleration for deep neural networks: The principles, progress, and challenges. IEEE Signal Processing Magazine (2018)
21. Chien, M.J., Kuh, E.: Solving nonlinear resistive networks using piecewise-linear analysis and simplicial subdivision. IEEE Transactions on Circuits and Systems (1977)
22. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems (1989)
23. De Palma, A., Behl, H., Bunel, R.R., Torr, P., Kumar, M.P.: Scaling the convex barrier with active sets. In: ICLR (2021)
24. Delarue, A., Anderson, R., Tjandraatmadja, C.: Reinforcement learning with combinatorial actions: An application to vehicle routing. In: NeurIPS (2020)
25. Denil, M., Shakibi, B., Dinh, L., Ranzato, M., Freitas, N.: Predicting parameters in deep learning. In: NeurIPS (2013)
26. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: NAACL (2019)
27. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: ICLR (2021)
28. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward networks. In: NFM (2018)
29. Ebrahimi, A., Klabjan, D.: Neuron-based pruning of deep neural networks with better generalization using kronecker factored curvature approximation. In: IJCNN (2023)
30. ElAraby, M., Wolf, G., Carvalho, M.: OAMIP: Optimizing ANN architectures using mixed-integer programming. In: CPAIOR (2023)
31. Fajemisin, A., Maragno, D., den Hertog, D.: Optimization with constraint learning: A framework and survey. European Journal of Operational Research (2023)
32. Feijoo, B., Meyer, R.R.: Piecewise-linear approximation methods for nonseparable convex optimization. Management Science (1988)
33. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints (2018)
34. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: ICLR (2019)
35. Frantar, E., Alistarh, D.: SparseGPT: Massive language models can be accurately pruned in one-shot. In: ICML (2023)
36. Funahashi, K.I.: On the approximate realization of continuous mappings by neural networks. Neural Networks (1989)
37. Ganev, I., Walters, R.: Model compression via symmetries of the parameter space (2022), https://openreview.net/forum?id=8MN_GH4Ckp4
38. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: AISTATS (2011)
39. Good, A., Lin, J., Sieg, H., Ferguson, M., Yu, X., Zhe, S., Wieczorek, J., Serra, T.: Recall distortion in neural network pruning and the undecayed pruning algorithm. In: NeurIPS (2022)

40. Grigsby, J.E., Lindsey, K.: On transversality of bent hyperplane arrangements and the topological expressiveness of ReLU neural networks. SIAM Journal on Applied Algebra and Geometry (2022)
41. Grimstad, B., Andersson, H.: ReLU networks as surrogate models in mixed-integer linear programs. Computers & Chemical Engineering (2019)
42. Gurobi Optimization: Gurobi Machine Learning. https://github.com/Gurobi/gurobi-machinelearning (2025), accessed: 2025-02-09
43. Haase, C.A., Hertrich, C., Loho, G.: Lower bounds on the depth of integral ReLU neural networks via lattice polytopes. In: ICLR (2023)
44. Hahnloser, R., Sarpeshkar, R., Mahowald, M., Douglas, R., Seung, S.: Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature (2000)
45. Hanin, B., Rolnick, D.: Complexity of linear regions in deep networks. In: ICML (2019)
46. Hanin, B., Rolnick, D.: Deep ReLU networks have surprisingly few activation patterns. In: NeurIPS (2019)
47. Hanin, B., Sellke, M.: Approximating continuous functions by ReLU nets of minimal width. arXiv:1710.11278 (2017)
48. Hanson, S., Pratt, L.: Comparing biases for minimal network construction with back-propagation. In: NeurIPS (1988)
49. Hassibi, B., Stork, D.: Second order derivatives for network pruning: Optimal Brain Surgeon. In: NeurIPS (1992)
50. Hassibi, B., Stork, D., Wolff, G.: Optimal brain surgeon and general network pruning. In: IEEE International Conference on Neural Networks (1993)
51. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: ICCV (2017)
52. Hendrycks, D., Gimpel, K.: Gaussian error linear units (GELUs). arXiv:1606.08415 (2016)
53. Hertrich, C., Basu, A., Summa, M.D., Skutella, M.: Towards lower bounds on the depth of ReLU neural networks. In: NeurIPS (2021)
54. Hertrich, C., Loho, G.: Neural networks and (virtual) extended formulations. arXiv:2411.03006 (2024)
55. Hinz, P., van de Geer, S.: A framework for the construction of upper bounds on the number of affine linear regions of ReLU feed-forward neural networks. IEEE Transactions on Information Theory (2019)
56. Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., Peste, A.: Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. Journal of Machine Learning Research (2021)
57. Hojny, C., Zhang, S., Campos, J.S., Misener, R.: Verifying message-passing neural networks via topology-based bounds tightening. In: ICML (2024)
58. Hooker, S., Courville, A., Clark, G., Dauphin, Y., Frome, A.: What do compressed deep neural networks forget? arXiv:1911.05248 (2019)
59. Hooker, S., Moorosi, N., Clark, G., Bengio, S., Denton, E.: Characterising bias in compressed models. arXiv:2010.03058 (2020)
60. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks (1989)
61. Huchette, J.: Advanced mixed-integer programming formulations: Methodology, computation, and application. Ph.D. thesis, Massachusetts Institute of Technology (2018)

62. Huchette, J., Muñoz, G., Serra, T., Tsay, C.: When deep learning meets polyhedral theory: A survey. arXiv:2305.00241 (2023)
63. Huchette, J., Vielma, J.P.: Nonconvex piecewise linear functions: Advanced formulations and simple modeling tools. Operations Research (2023)
64. Janowsky, S.: Pruning versus clipping in neural networks. Physical Review A (1989)
65. Jin, T., Roy, D., Carbin, M., Frankle, J., Dziugaite, G.: On neural network pruning's effect on generalization. In: NeurIPS (2022)
66. Kanamori, K., Takagi, T., Kobayashi, K., Ike, Y., Uemura, K., Arimura, H.: Ordered counterfactual explanation by mixed-integer linear optimization. In: AAAI (2021)
67. Kumar, A., Serra, T., Ramalingam, S.: Equivalent and approximate transformations of deep neural networks. arXiv:1905.11428 (2019)
68. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998)
69. LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. In: NeurIPS (1989)
70. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature (2015)
71. Lee, N., Ajanthan, T., Torr, P.: SNIP: Single-shot network pruning based on connection sensitivity. In: ICLR (2019)
72. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: NeurIPS (2018)
73. Liebenwein, L., Baykal, C., Carter, B., Gifford, D., Rus, D.: Lost in pruning: The effects of pruning neural networks beyond test accuracy. In: MLSys (2021)
74. 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 (2021)
75. Liu, X., Han, X., Zhang, N., Liu, Q.: Certified monotonic neural networks. In: NeurIPS. vol. 33 (2020)
76. Liu, X., Dvorkin, V.: Optimization over trained neural networks: Difference-of-convex algorithm and application to data center scheduling. IEEE Control Systems Letters (2025)
77. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward ReLU neural networks. arXiv:1706.07351 (2017)
78. Lueg, L., Grimstad, B., Mitsos, A., Schweidtmann, A.M.: reluMIP: Open source tool for MILP optimization of relu neural networks (2021). https://doi.org/https://doi.org/10.5281/zenodo.5601907, https://github.com/ChemEngAI/ReLU_ANN_MILP
79. Luo, J., Wu, J., Lin, W.: ThiNet: A filter level pruning method for deep neural network compression. In: ICCV (2017)
80. Ma, X., Fang, G., Wang, X.: LLM-Pruner: On the structural pruning of large language models. In: NeurIPS (2023)
81. Magnani, A., Boyd, S.P.: Convex piecewise-linear fitting. Optimization Engineering (2009)
82. Mangasarian, O.L., Rosen, J.B., Thompson, M.E.: Global minimization via piecewise-linear underestimation. Journal of Global Optimization (2005)
83. Maragno, D., Wiberg, H., Bertsimas, D., Birbil, S.I., Hertog, D.d., Fajemisin, A.: Mixed-integer optimization with constraint learning. Operations Research (2023)
84. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part I — convex underestimating problems. Mathematical Programming (1976)

85. McDonald, T., Tsay, C., Schweidtmann, A.M., Yorke-Smith, N.: Mixed-integer optimisation of graph neural networks for computer-aided molecular design. Computers & Chemical Engineering (2024)
86. Misener, R., Floudas, C.A.: Piecewise-linear approximations of multidimensional functions. Journal of Optimization Theory and Applications (2010)
87. Misener, R., Biegler, L.: Formulating data-driven surrogate models for process optimization. Computers & Chemical Engineering (2023)
88. Montúfar, G.: Notes on the number of linear regions of deep neural networks. In: Sampling Theory and Applications (SampTA) (2017)
89. Montúfar, G., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. In: NeurIPS. vol. 27 (2014)
90. Montúfar, G., Ren, Y., Zhang, L.: Sharp bounds for the number of regions of maxout networks and vertices of Minkowski sums. SIAM Journal on Applied Algebra and Geometry (2022)
91. Mozer, M., Smolensky, P.: Using relevance to reduce network size automatically. Connection Science (1989)
92. 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)
93. Nair, V., Hinton, G.: Rectified linear units improve restricted boltzmann machines. In: ICML (2010)
94. Paganini, M.: Prune responsibly. arXiv:2009.09936 (2020)
95. Pascanu, R., Montúfar, G., Bengio, Y.: On the number of response regions of deep feedforward networks with piecewise linear activations. In: ICLR (2014)
96. Perakis, G., Tsiourvas, A.: Optimizing objective functions from trained ReLU neural networks via sampling. arXiv:2205.14189 (2022)
97. Phuong, M., Lampert, C.H.: Functional vs. parametric equivalence of ReLU networks. In: ICLR (2020)
98. Plate, C., Hahn, M., Klimek, A., Ganzer, C., Sundmacher, K., Sager, S.: An analysis of optimization problems involving ReLU neural networks. arXiv:2502.03016 (2025)
99. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/ language_understanding_paper.pdf (2018), accessed: 2025-03-04
100. Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., Dickstein, J.: On the expressive power of deep neural networks. In: ICML (2017)
101. Riera, C., Rey, C., Serra, T., Puertas, E., Pujol, O.: Training thinner and deeper neural networks: Jumpstart regularization. In: CPAIOR (2022)
102. Rolnick, D., Kording, K.: Reverse-engineering deep ReLU networks. In: International Conference on Machine Learning (ICML) (2020)
103. Rosen, J.B., Pardalos, P.M.: Global minimization of large-scale constrained concave quadratic problems by separable programming. Mathematical Programming (1986)
104. Rössig, A., Petkovic, M.: Advances in verification of ReLU neural networks. Journal of Global Optimization (2021)
105. Safran, I., Reichman, D., Valiant, P.: How many neurons does it take to approximate the maximum? In: SODA (2024)
106. Say, B., Wu, G., Zhou, Y.Q., Sanner, S.: Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In: IJCAI (2017)

107. Schweidtmann, A.M., Mitsos, A.: Deterministic global optimization with artificial neural networks embedded. Journal of Optimization Theory and Applications (2019)
108. Serra, T., Ramalingam, S.: Empirical bounds on linear regions of deep rectifier networks. In: AAAI (2020)
109. Serra, T., Tjandraatmadja, C., Ramalingam, S.: Bounding and counting linear regions of deep neural networks. In: ICML (2018)
110. Serra, T., Yu, X., Kumar, A., Ramalingam, S.: Scaling up exact neural network compression by ReLU stability. In: NeurIPS (2021)
111. Serra, T., Kumar, A., Ramalingam, S.: Lossless compression of deep neural networks. In: CPAIOR (2020)
112. Shi, C., Emadikhiav, M., Lozano, L., Bergman, D.: Constraint learning to define trust regions in optimization over pre-trained predictive models. INFORMS Journal on Computing (2024)
113. Singh, S.P., Alistarh, D.: WoodFisher: Efficient second-order approximation for neural network compression. In: NeurIPS (2020)
114. Sosnin, P., Tsay, C.: Scaling mixed-integer programming for certification of neural network controllers using bounds tightening. In: CDC (2024)
115. Sourek, G., Zelezny, F.: Lossless compression of structured convolutional models via lifting. In: ICLR (2021)
116. 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)
117. Sun, M., Liu, Z., Bair, A., Kolter, J.Z.: A simple and effective pruning approach for large language models. In: ICLR (2024)
118. Sun, R., Li, D., Liang, S., Ding, T., Srikant, R.: The global landscape of neural networks: An overview. IEEE Signal Processing Magazine **37** (2020)
119. Telgarsky, M.: Representation benefits of deep feedforward networks. arXiv:1509.08101 (2015)
120. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: ICLR (2019)
121. Tong, J., Cai, J., Serra, T.: Optimization over trained neural networks: Taking a relaxing walk. In: CPAIOR (2024)
122. Tran, C., Fioretto, F., Kim, J.E., Naidu, R.: Pruning has a disparate impact on model accuracy. In: NeurIPS (2022)
123. Tsay, C., Kronqvist, J., Thebelt, A., Misener, R.: Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. In: NeurIPS (2021)
124. Tseran, H., Montúfar, G.: On the expected complexity of maxout networks. In: NeurIPS (2021)
125. Tsiourvas, A., Sun, W., Perakis, G.: Manifold-aligned counterfactual explanations for neural networks. In: AISTATS (2024)
126. Turner, M., Chmiela, A., Koch, T., Winkler, M.: SurrogateLIB: An extendable library of mixed- integer programs with embedded machine learning predictors (May 2024), https://doi.org/10.5281/zenodo.11231147
127. Turner, M., Chmiela, A., Koch, T., Winkler, M.: PySCIPOpt-ML: Embedding trained machine learning models into mixed-integer programs. In: CPAIOR (2025)
128. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: NeurIPS (2017)
129. Verma, S., Pesquet, J.C.: Sparsifying networks via subdifferential inclusion. In: ICML (2021)

130. Vielma, J.P., Ahmed, S., Nemhauser, G.: Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. Operations Research (2010)
131. Wang, K., Lozano, L., Cardonha, C., Bergman, D.: Optimizing over an ensemble of trained neural networks. INFORMS Journal on Computing (2023)
132. Wang, Y.: Estimation and comparison of linear regions for ReLU networks. In: IJCAI (2022)
133. Williams, H.P.: Model Building in Mathematical Programming. Wiley, 5 edn. (2013)
134. Wu, D., Modoranu, I.V., Safaryan, M., Kuznedelev, D., Alistarh, D.: The iterative optimal brain surgeon: Faster sparse recovery by leveraging second-order information. In: NeurIPS (2024)
135. Wu, G., Say, B., Sanner, S.: Scalable planning with deep neural network learned transition models. Journal of Artificial Intelligence Research (2020)
136. Xia, M., Gao, T., Zeng, Z., Chen, D.: Sheared LLaMA: Accelerating language model pre-training via structured pruning. In: ICLR (2024)
137. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv **1708.07747** (2017)
138. Xiao, K.Y., Tjeng, V., Shafiullah, N.M., Madry, A.: Training for faster adversarial robustness verification via inducing ReLU stability. In: ICLR (2019)
139. Yang, S., Bequette, B.W.: Optimization-based control using input convex neural networks. Computers & Chemical Engineering (2021)
140. Yarotsky, D.: Error bounds for approximations with deep ReLU networks. Neural Networks (2017)
141. Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., Liu, Q.: Good subnetworks provably exist: Pruning via greedy forward selection. In: ICML (2020)
142. Yu, X., Serra, T., Ramalingam, S., Zhe, S.: The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In: ICML (2022)
143. Zhao, H., Hijazi, H., Jones, H., Moore, J., Tanneau, M., Hentenryck, P.V.: Bound tightening using rolling-horizon decomposition for neural network verification. In: CPAIOR (2024)
144. Zhu, Y., Burer, S.: An extended validity domain for constraint learning. arXiv:2406.10065 (2024)