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

Hung Pham[a], Aiden Ren[a], Ibrahim Tahir[a], Jiatai Tong[b], Thiago Serra[c,*]

[a] *Bucknell University, Lewisburg, PA, United States*
[b] *Northwestern University, Evanston, IL, United States*
[c] *University of Iowa, Iowa City, IA, United States*

Abstract

We can approximate a constraint or an objective function that is uncertain or nonlinear with a neural network that we embed in the optimization model. This approach, which is known as *constraint learning*, faces the challenge that optimization models with neural network surrogates are harder to solve. Such difficulties have motivated studies on model reformulation, specialized optimization algorithms, and—to a lesser extent—pruning of the embedded networks. In this work, we double down on the use of surrogates by applying network pruning to produce a surrogate of the neural network itself. In the context of using a Mixed-Integer Linear Programming (MILP) solver to verify neural networks, we obtained faster adversarial perturbations for dense neural networks by using sparse surrogates, especially—and surprisingly—if not taking the time to finetune the sparse network to make up for the loss in accuracy. In other words, we show that a pruned network with bad classification performance can still be a good—and more efficient—surrogate.

Keywords: constraint learning, mixed-integer programming, neural network pruning, neural network verification, piecewise linear approximation

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

*Corresponding author
Email address: thiago-serra@uiowa.edu (Thiago Serra)

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 compeling circumstance is when the formulation is intractable for a combination of factors such as being nonlinear, nonconvex, and very large [1–4].

Those possibilities sparked several studies in terms of modeling. The applications considered include scholarship allocation [5], chemotherapy [6], molecular design [7], power grid operation [8–10], and automated control in general [11–14]. We can also use such embeddings to evaluate the neural network itself for adversarial perturbations [15–19], compression [20–22], counterfactual explanations [23, 24], equivalence [25], expressiveness [26–28], monotonicity [29], and reachability [30, 31]. Conversely, when the neural network is trained for reinforcement learning, the constraints in the optimization model can be used for limiting the action space [32, 33]. Many frameworks have been proposed to embed neural networks and other machine learning models as part of optimization models, such as JANOS [5], reluMIP [34], OMLT [35], OCL [36], OptiCL [6], Gurobi Machine Learning [37], and PySCIPOpt-ML [38].

Some neural network architectures are more convenient to embed in optimization models. For example, we typically use the Rectified Linear Unit (ReLU) activation function [39, 40] 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 [40–42], being later shown that ReLU networks are universal function approximators [43, 44]. More recently, Gaussian Error Linear Units (GELUs) [45] have been used from the very beginning in many foundation models based on the transformer architecture [46], such as GPT [47], BERT [48], and ViT [49]. 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 [50]. In the context of nonlinear optimization, substantial work has been focused on using piecewise linear approximations as optimization model surrogates [51–61] 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 [26–28, 50, 62–81].

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 prohibitivelly large [82], making it difficult to use an off-the-self MILP solver. Hence, many studies have focused on how to tackle constraint learning models. Such studies ranged from model reformulation [10, 82–84] to methods for generating big M coefficients with activation bounds [14, 15, 84–90] and parameter rescaling [91], activation inferences for fixing variables and limiting search space [27, 92–94], cutting planes [16], and heuristic solutions [95, 96]. Another—perhaps overlooked—approach is to try working with neural networks that are sparser [11, 93] and smaller [97].

This latter approach combines mathematical optimization and deep learning folklore. In mathematical optimization, we know that sparse optimization models tend to be solved much more efficiently in practice. In deep learning, we know that removing a large amount of parameters of a trained neural network can be compensated, in terms of preserving accuracy, with finetuning. Finetuning consists of a few extra steps of training, but considerably fewer than training the neural network from scratch, since the pruned network has latent information on the task. Hence, neural networks should be preferably sparse when used in constraint learning models [11, 93, 97].

In this work specifically, we consider a complicating factor: **what if the (dense) neural network is a given?** We assume as a corollary that the approximation of a constraint learning model using a sparse network can be solved more efficiently. That can be convenient in applications such as neural network verification, in which we do not need to find the optimal solution necessarily. However, it is possible that the time finetuning the pruned neural network for recovering its accuracy may offset the gains from solving a sparser model. From the premise that the latent information may suffice, we further posit that finetuning is not necessary to use a pruned network as a surrogate. Hence, our hypothesis is that we can replace the (dense) neural network with its pruned and nonfinetuned counterpart in the optimization model, even if this pruned neural network is lacking in terms of accuracy, and then still find a solution that would be reasonably good for the original model. Based on our experiments, we found out that we can indeed obtain faster adversarial inputs to (dense) neural networks by using their sparse surrogates.

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 [98], since the largest output of softmax matches the largest input of softmax, hence not being necessary to include softmax to find adversarial inputs to a neural network.

Each neural network has an input $\boldsymbol{x} = [x_1 \; x_2 \; \ldots \; x_{n_0}]^\top$ from a bounded domain $\mathbb{X}$ and corresponding output $\boldsymbol{y} = [y_1 \; y_2 \; \ldots \; y_m]^\top$, and each layer $l \in \mathbb{L} = \{1, 2, \ldots, L\}$ has output $\boldsymbol{h}^l = [h_1^l \; h_2^l \ldots h_{n_l}^l]^\top$ from neurons indexed by $i \in \mathbb{N}_l = \{1, 2, \ldots, n_l\}$. 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 $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i \in \mathbb{S}}$ in the training set.

When optimizing over the trained neural network, 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 set of parameters $\{(\boldsymbol{W}^l, \boldsymbol{b}^l)\}_{l \in \mathbb{L}}$ fixed by training. For each layer $l \in \mathbb{L}$ and neuron $i \in \mathbb{N}_l$, we use a binary variable $\boldsymbol{z}_i^l$ to map inputs to outputs using MILP:

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

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

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

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

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

The indicator constraints (2)–(3) can be modeled with big M constraints [99].

When neural networks are used for classification, one application of MILP is to determine if there is an adversarial perturbation for a given input $\boldsymbol{x}^{(i)}, i \in \mathbb{S}$ from the training set. If this input is correctly classified with a 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., $y_{j'} > y_j$.[1] That leads to the following MILP model:

$$\max \quad y_{j'} - y_j \tag{6}$$

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

$$\sum_{k \in \{1,\dots,n_0\}} |\boldsymbol{x}_k - \boldsymbol{x}_k^{(i)}| \leq \varepsilon \tag{8}$$

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.

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 to obtain comparable variants that are smaller and faster [100–103].

We will explore how that can help tackling constraint learning models.

3.1. Background

Neural networks are pruned by either (i) removing connections, which is equivalent to zeroing out specific parameters (*unstructured* pruning); or (ii) removing units, such as neurons, convolutional filters, or layers (*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 [104].

But why do we need network pruning? A larger neural network has a smoother loss landscape [105, 106], which facilitates training convergence; and the larger size may also prevent layers from becoming inactive [107],

[1] 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., $y_{j''} > y_{j'} > y_j$.

which is a common cause for unsuccessful training. **Why does network pruning work?** In larger networks, there is redundancy among the paramenters [108], and zeroing out parameters leads to a loss landscape from which finetuning the pruned network for recovering the original performance converges considerably faster than training [109]. From a model flexibility perspective, unstructured pruning at moderate rates has little effect on the expressiveness of the neural network architecture [28]. **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 [110]. However, that varies with the task for which the network is trained [111]. Moreover, pruning may have a disparate effect across classes [112–114], which may lead to pruned networks that exacerbate existing performance differences [115]. On the bright side, a smaller amount of pruning may actually correct such distortions [116]. **What should we prune?** The two main philosophies [117] are (1) to remove parameters with the smallest absolute value (dating back to [118–120]); and (2) to remove parameters with the smallest expected impact on the output (dating back to [121–123], and including the special case of exact compression [20, 21, 124, 125]). **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 [126], or even before training [127].

Mathematical optimization has been extensively used in more sophisticated pruning methods [20–22, 28, 121–123, 128–138]. 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, or *Magnitude Pruning* (MP) [118–120], remains "unreasonably effective" [134].

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. [11] observed that a modest amount of unstructured pruning—removing about 20% of the parameters—significantly reduced the runtime for solving the optimization model. More recently, Cacciola et al. [97] 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.

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 $\mathcal{S}$

1: **while** trying to solve $\mathbf{VNN}(\mathcal{S}, \varepsilon, x^{(i)}, j, j')$ **do** ▷ MILP solver call
2: **if** solution $(\boldsymbol{x}, \boldsymbol{y}^S)$ found **then** ▷ Feasible solution callback
3: $\boldsymbol{y}^D = \mathcal{D}(\boldsymbol{x})$ ▷ Output of the dense model $\mathcal{D}$
4: **if** $y^D_{j'} > y^D_j$ **then** ▷ Check if $\boldsymbol{x}$ is adversarial to $\mathcal{D}$
5: **return** $\boldsymbol{x}$ ▷ Adversarial input found
6: **end if**
7: **end if**
8: **end while**
9: **return** $\emptyset$ ▷ No adversarial input found

3.2. The Sparse Surrogate Approach

Now we succinctly describe how we use a pruned neural network to obtain solutions for a constraint learning model. In particular, we consider the case of a network verification problem, in which we validate if an adversarial input to the pruned network is also an adversarial input to the original network.

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 of those networks for input $\boldsymbol{x}$. Moreover, let $\text{VNN}(\mathcal{N}, \boldsymbol{x}^{(i)}, \varepsilon, j, j')$ be the MILP formulation (6)–(8) for a verification problem on neural network $\mathcal{N} \in \{\mathcal{D}, \mathcal{S}\}$ starting from input $\boldsymbol{x}^{(i)}$ and with a 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 would be sufficient to find $\boldsymbol{x}$ such that $y^N_{j'} > y^N_j$.

In order to obtain a solution with positive value for the verification problem using model $\mathbf{VNN}(\mathcal{D}, \varepsilon, x^{(i)}, j, j')$, we resort to solving the sparse model $\mathbf{VNN}(\mathcal{S}, \varepsilon, x^{(i)}, j, j')$ as outlined in Algorithm 1. Line 1 is flexible enough to accommodate solving the sparse model to optimality or stopping after a certain time limit. For every solution found for the sparse model in Line 2, we obtain the corresponding output for the (dense) network $\mathcal{D}$ in Line 3 and then evaluate if that would be an adversarial input to $\mathcal{D}$ in Line 4. Due to the differences between networks $\mathcal{S}$ and $\mathcal{D}$, it is possible that we may find not an adversarial input to $\mathcal{D}$ even if one exists and we solve the sparse model to optimality. However, we report in the next section that we often do find such an input—and faster than trying to solve the dense model directly.

4. Experiments

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 [139] as the basis for training neural networks and producing network verification problems, starting with the MNIST dataset [140] and then extending the code to also work with the Fashion-MNIST dataset [141]. 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 solved 6,560 verification problems.

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) for solving the verification model directly (x axis) and indirectly (y axis) in the plots of Figure 1 for MNIST and of Figure 2 for Fashion-MNIST. Each row corresponds to a different pruning rate. The first column shows the results for pruned networks that are not finetuned, the second column shows the results for pruned networks that are finetuned, and the third column adds the finetuning time to the runtime of the latter. 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% when some instances time out for both.

We start our analysis by enumerating some observations about the plots:

(i) We found adversarial inputs faster for most instances regardless of pruning rate and of the pruned neural networks being finetuned or not.

(ii) When there are timeouts (i.e., not finding an adversarial input) for solving the verification problem directly (as with MNIST), then using our approach with a small pruning rate reduces the number of timeouts.

(iii) The number of runtime improvements increases with pruning rate up to a peak (90% for both datasets, finetuned or not) and then decreases.

(iv) With greater pruning rates, the differences between runtimes become more extreme in our favor, but at the same time the timeouts increase.

(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) For the lowest pruning rates (one for MNIST and three for Fashion-MNIST), the results were better without finetuning.

(vii) For the higher pruning rates except the highest, the difference in the percentage of instances solved faster with finetuning is only on the second most significant digit (e.g., 98.8% instead of 93.5% on MNIST and 93.8% instead of 91.2% on Fashion-MNIST for pruning rate 90%).

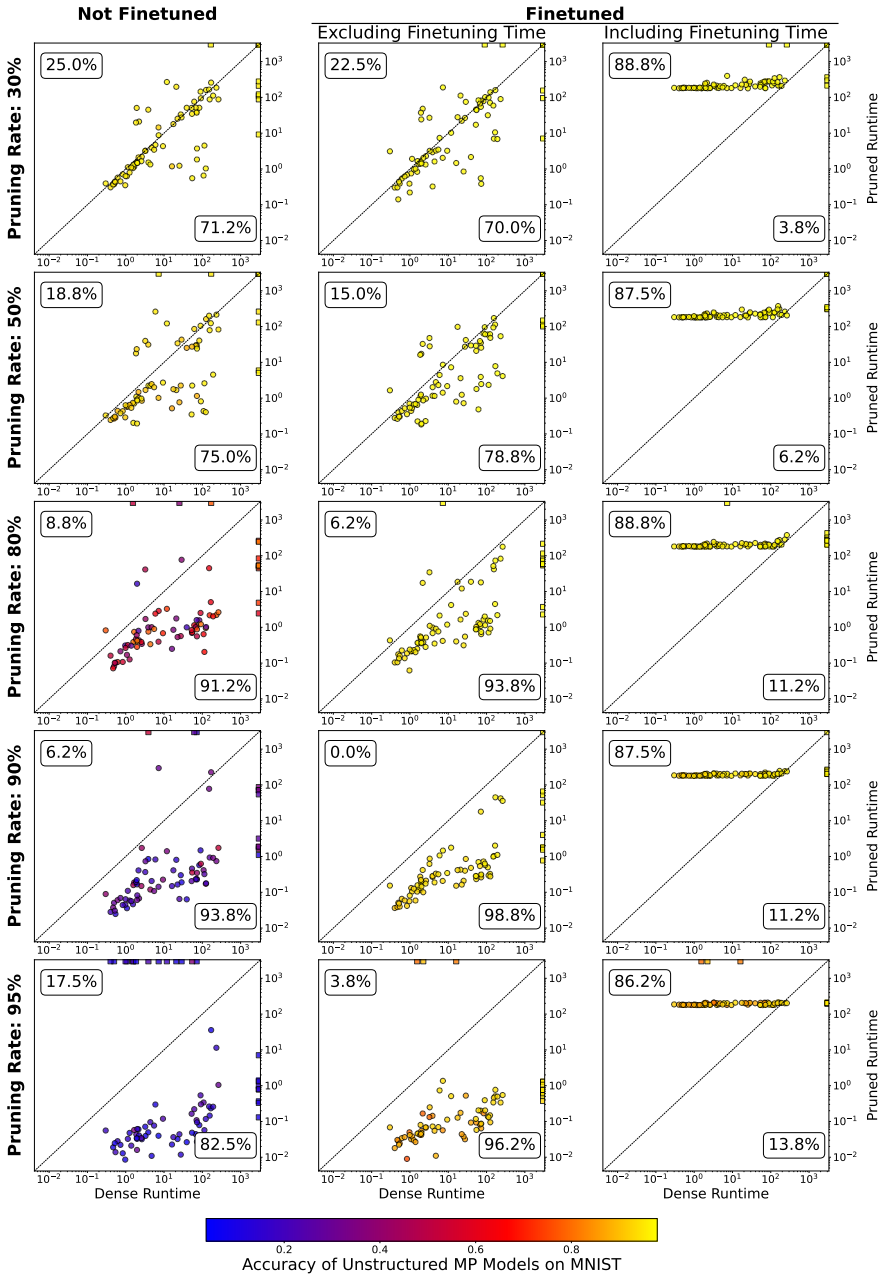

Figure 1: Time for finding adversarial inputs to neural networks trained on MNIST by solving the verification problem directly (x axis) or indirectly with Algorithm 1 (y axis). Squares on top or (and) right sides indicate no adversarial input found for either (both).

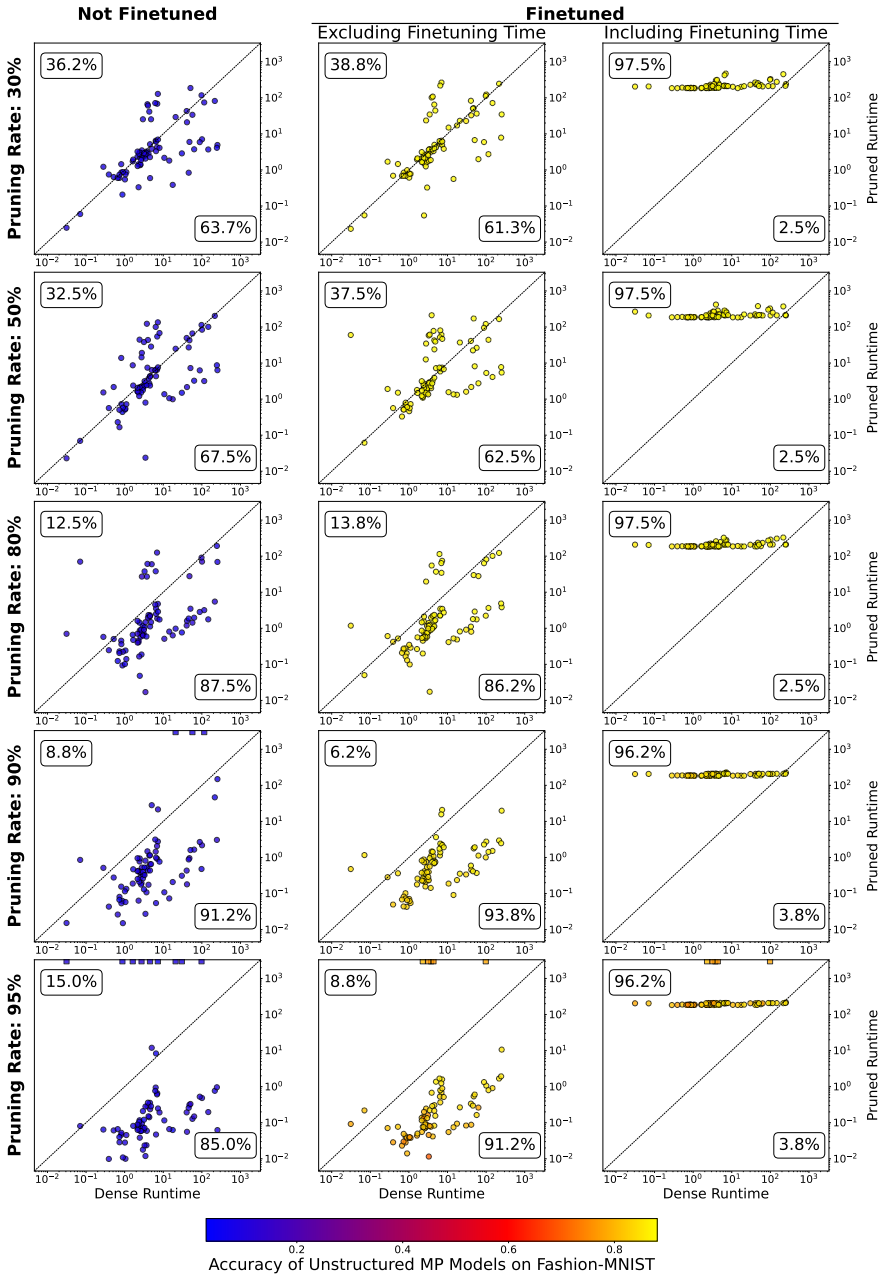

Figure 2: Time for finding adversarial inputs to networks trained on Fashion-MNIST by solving the verification problem directly (x axis) or indirectly with Algorithm 1 (y axis). Squares on top or (and) right sides indicate no adversarial input found for either (both).

(viii) If we account for the cost of finetuning, then it is generally faster to solve the verification problem directly instead of using a finetuned network.

Based on those observations, we draw the following conclusions:

(I) Using our approach in network verification is advantageous in terms of individual runtimes (i) as well as number of instances solved (ii).

(II) The pruning rate can be adjusted for different purposes, from finding more adversarial inputs up to a time limit at lower rates (ii) to finding most adversarial inputs faster at higher rates (iii) and fewer adversarial inputs but in a much shorter amount of time at the highest rate (iv).

(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).

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

First, we considered the impact of other choices that we could have made on how to prune the neural networks. Figure 3 compares the joint results on MNIST and Fashion-MNIST by using unstructured MP as before (first column), then by replacing only unstructured with structured pruning (second column), and then by replacing only MP with RP (third column). Considering possible questions due to the favorable results for structured pruning over unstructured pruning in some of the plots, Table 1 summarizes the number of instances on both datasets in which an adversarial input would have been

		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%

Table 1: Percentage of instances for which solving the verification problem indirectly on a pruned neural network is faster by pruning rate (columns); using unstructured and structured pruning (top and bottom rows); and not finetuning (NF) or finetuning (F).

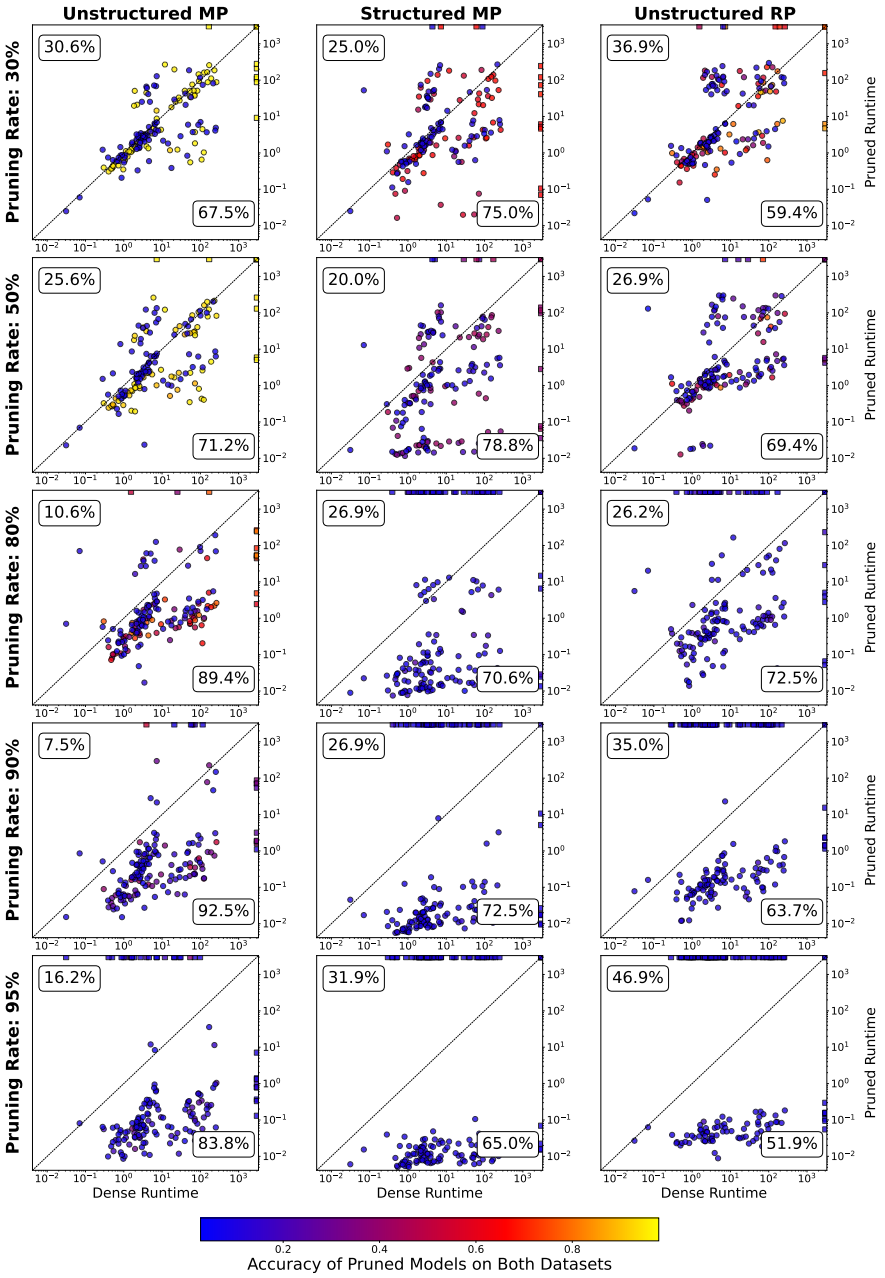

Figure 3: Time for finding adversarial inputs to neural networks trained on MNIST and Fashion-MNIST by solving the verification problem directly (x axis) or indirectly with Algorithm 1 (y axis). The columns represent different forms of pruning the neural networks. Squares on top or (and) right sides indicate no adversarial input found for either (both).

found faster than directly solving the verification problem by using unstructured or structured network pruning with or without 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. In contrast, 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 directly solving the verification problem according to the size of the input, number of layers (network depth), and neurons per layer (layer width).

We enumerate additional observations based on the extra plots and tables:

(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 the pruning rate. The opposite happens with structured pruning.

(xii) The results for unstructured MP are better than those for unstructured RP, and the difference grows with the pruning rate.

		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%

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

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

We draw additional conclusions based on the 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).

(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).

5. Conclusion

In this work, we proposed replacing a (dense) neural network embedded in an optimization model with a pruned version of that same neural network. These are models that become very difficult to solve as the neural networks become larger, but for which we may not need to find an optimal solution. By making the models sparser, we naturally expect to solve them faster.

Our study case was the most popular type of optimization problem involving a trained neural network, which is verifying if there are adversarial perturbations to particular inputs. With the goal of solving the optimization problem embedding a dense neural network, we tackled that problem indirectly through a drastically sparsified neural network that serves as a surrogate. Even if the sparsified neural network had very low accuracy, we were able to obtain adversarial inputs in most of the instances tested—and often considerably faster than by solving the verification problem directly.

With our experiments, we have learned that a cost-effective approach consists of applying unstructured pruning while carefully choosing which connections to prune but not finetuning the pruned network afterwards.

We believe that this work contributes to understanding how to tackle such optimization models with embedded neural networks more effectively.

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

References

[1] R. Misener, L. Biegler, Formulating data-driven surrogate models for process optimization, Computers & Chemical Engineering (2023).

[2] K. Wang, L. Lozano, C. Cardonha, D. Bergman, Optimizing over an ensemble of trained neural networks, INFORMS Journal on Computing (2023).

[3] C. Shi, M. Emadikhiav, L. Lozano, D. Bergman, Constraint learning to define trust regions in optimization over pre-trained predictive models, INFORMS Journal on Computing (2024).

[4] Y. Zhu, S. Burer, An extended validity domain for constraint learning, arXiv:2406.10065 (2024).

[5] D. Bergman, T. Huang, P. Brooks, A. Lodi, A. U. Raghunathan, JANOS: An integrated predictive and prescriptive modeling framework, INFORMS Journal on Computing (2022).

[6] D. Maragno, H. Wiberg, D. Bertsimas, S. I. Birbil, D. d. Hertog, A. Fajemisin, Mixed-integer optimization with constraint learning, Operations Research (2023).

[7] T. McDonald, C. Tsay, A. M. Schweidtmann, N. Yorke-Smith, Mixed-integer optimisation of graph neural networks for computer-aided molecular design, Computers & Chemical Engineering (2024).

[8] Y. Chen, Y. Shi, B. Zhang, Data-driven optimal voltage regulation using input convex neural networks, Electric Power Systems Research (2020).

[9] I. Murzakhanov, A. Venzke, G. S. Misyris, S. Chatzivasileiadis, Neural networks for encoding dynamic security-constrained optimal power flow, in: Bulk Power Systems Dynamics and Control Sympositum, 2022.

[10] X. Liu, V. Dvorkin, Optimization over trained neural networks: Difference-of-convex algorithm and application to data center scheduling, arXiv:2503.17506 (2025).

[11] B. Say, G. Wu, Y. Q. Zhou, S. Sanner, Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming, in: International Joint Conference on Artificial Intelligence (IJCAI), 2017.

[12] G. Wu, B. Say, S. Sanner, Scalable planning with deep neural network learned transition models, Journal of Artificial Intelligence Research (2020).

[13] S. Yang, B. W. Bequette, Optimization-based control using input convex neural networks, Computers & Chemical Engineering (2021).

[14] P. Sosnin, C. Tsay, Scaling mixed-integer programming for certification of neural network controllers using bounds tightening, in: CDC, 2024.

[15] C. Cheng, G. Nührenberg, H. Ruess, Maximum resilience of artificial neural networks, in: ATVA, 2017.

[16] R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, J. P. Vielma, Strong mixed-integer programming formulations for trained neural networks, Mathematical Programming (2020).

[17] A. Rössig, M. Petkovic, Advances in verification of ReLU neural networks, Journal of Global Optimization (2021).

[18] C. A. Strong, H. Wu, A. Zeljić, K. D. Julian, G. Katz, C. Barrett, M. J. Kochenderfer, Global optimization of objective functions represented by ReLU networks, Machine Learning (2021).

[19] A. De Palma, H. Behl, R. R. Bunel, P. Torr, M. P. Kumar, Scaling the convex barrier with active sets, in: ICLR, 2021.

[20] T. Serra, A. Kumar, S. Ramalingam, Lossless compression of deep neural networks, in: CPAIOR, 2020.

[21] T. Serra, X. Yu, A. Kumar, S. Ramalingam, Scaling up exact neural network compression by ReLU stability, in: NeurIPS, 2021.

[22] M. ElAraby, G. Wolf, M. Carvalho, OAMIP: Optimizing ANN architectures using mixed-integer programming, in: CPAIOR, 2023.

[23] K. Kanamori, T. Takagi, K. Kobayashi, Y. Ike, K. Uemura, H. Arimura, Ordered counterfactual explanation by mixed-integer linear optimization, in: AAAI, 2021.

[24] A. Tsiourvas, W. Sun, G. Perakis, Manifold-aligned counterfactual explanations for neural networks, in: AISTATS, 2024.

[25] A. Kumar, T. Serra, S. Ramalingam, Equivalent and approximate transformations of deep neural networks, arXiv:1905.11428 (2019).

[26] T. Serra, C. Tjandraatmadja, S. Ramalingam, Bounding and counting linear regions of deep neural networks, in: ICML, 2018.

[27] T. Serra, S. Ramalingam, Empirical bounds on linear regions of deep rectifier networks, in: AAAI, 2020.

[28] J. Cai, K.-N. Nguyen, N. Shrestha, A. Good, R. Tu, X. Yu, S. Zhe, T. Serra, Getting away with more network pruning: From sparsity to geometry and linear regions, in: CPAIOR, 2023.

[29] X. Liu, X. Han, N. Zhang, Q. Liu, Certified monotonic neural networks, in: NeurIPS, volume 33, 2020.

[30] A. Lomuscio, L. Maganti, An approach to reachability analysis for feed-forward ReLU neural networks, arXiv:1706.07351 (2017).

[31] S. Dutta, S. Jha, S. Sankaranarayanan, A. Tiwari, Output range analysis for deep feedforward networks, in: NFM, 2018.

[32] A. Delarue, R. Anderson, C. Tjandraatmadja, Reinforcement learning with combinatorial actions: An application to vehicle routing, in: NeurIPS, 2020.

[33] R.-A. Burtea, C. Tsay, Safe deployment of reinforcement learning using deterministic optimization over neural networks, Computer Aided Chemical Engineering (2023).

[34] L. Lueg, B. Grimstad, A. Mitsos, A. M. Schweidtmann, reluMIP: Open source tool for MILP optimization of relu neural networks, 2021. URL: `https://github.com/ChemEngAI/ReLU_ANN_MILP`. doi:https://doi.org/10.5281/zenodo.5601907.

[35] F. Ceccon, J. Jalving, J. Haddad, A. Thebelt, C. Tsay, C. D. Laird, R. Misener, Omlt: Optimization & machine learning toolkit, Journal of Machine Learning Research 23 (2022) 1–8.

[36] A. Fajemisin, D. Maragno, D. den Hertog, Optimization with constraint learning: A framework and survey, European Journal of Operational Research (2023).

[37] Gurobi Optimization, Gurobi Machine Learning, `https://github.com/Gurobi/gurobi-machinelearning`, 2025. Accessed: 2025-02-09.

[38] M. Turner, A. Chmiela, T. Koch, M. Winkler, PySCIPOpt-ML: Embedding trained machine learning models into mixed-integer programs, arXiv:2312.08074 (2023).

[39] R. Hahnloser, R. Sarpeshkar, M. Mahowald, R. Douglas, S. Seung, Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit, Nature (2000).

[40] V. Nair, G. Hinton, Rectified linear units improve restricted boltzmann machines, in: ICML, 2010.

[41] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: AISTATS, 2011.

[42] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature (2015).

[43] D. Yarotsky, Error bounds for approximations with deep ReLU networks, Neural Networks (2017).

[44] B. Hanin, M. Sellke, Approximating continuous functions by ReLU nets of minimal width, arXiv:1710.11278 (2017).

[45] D. Hendrycks, K. Gimpel, Gaussian error linear units (GELUs), arXiv:1606.08415 (2016).

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: NeurIPS, 2017.

[47] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving language understanding by generative pre-training, https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf, 2018. Accessed: 2025-03-04.

[48] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: NAACL, 2019.

[49] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, An image is worth 16x16 words: Transformers for image recognition at scale, in: ICLR, 2021.

[50] R. Arora, A. Basu, P. Mianjy, A. Mukherjee, Understanding deep neural networks with rectified linear units, in: ICLR, 2018.

[51] G. P. McCormick, Computability of global solutions to factorable nonconvex programs: Part I — convex underestimating problems, Mathematical Programming (1976).

[52] M.-J. Chien, E. Kuh, Solving nonlinear resistive networks using piecewise-linear analysis and simplicial subdivision, IEEE Transactions on Circuits and Systems (1977).

[53] J. B. Rosen, P. M. Pardalos, Global minimization of large-scale constrained concave quadratic problems by separable programming, Mathematical Programming (1986).

[54] B. Feijoo, R. R. Meyer, Piecewise-linear approximation methods for nonseparable convex optimization, Management Science (1988).

[55] D. A. Babayev, Piece-wise linear approximation of functions of two variables, Journal of Heuristics (1997).

[56] O. L. Mangasarian, J. B. Rosen, M. E. Thompson, Global minimization via piecewise-linear underestimation, Journal of Global Optimization (2005).

[57] A. Magnani, S. P. Boyd, Convex piecewise-linear fitting, Optimization Engineering (2009).

[58] J. P. Vielma, S. Ahmed, G. Nemhauser, Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions, Operations Research (2010).

[59] R. Misener, C. A. Floudas, Piecewise-linear approximations of multidimensional functions, Journal of Optimization Theory and Applications (2010).

[60] H. P. Williams, Model Building in Mathematical Programming, 5 ed., Wiley, 2013.

[61] J. Huchette, J. P. Vielma, Nonconvex piecewise linear functions: Advanced formulations and simple modeling tools, Operations Research (2023).

[62] R. Pascanu, G. Montúfar, Y. Bengio, On the number of response regions of deep feedforward networks with piecewise linear activations, in: ICLR, 2014.

[63] G. Montúfar, R. Pascanu, K. Cho, Y. Bengio, On the number of linear regions of deep neural networks, in: NeurIPS, volume 27, 2014.

[64] M. Telgarsky, Representation benefits of deep feedforward networks, arXiv:1509.08101 (2015).

[65] G. Montúfar, Notes on the number of linear regions of deep neural networks, in: Sampling Theory and Applications (SampTA), 2017.

[66] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, J. Dickstein, On the expressive power of deep neural networks, in: ICML, 2017.

[67] P. Hinz, S. van de Geer, 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).

[68] B. Hanin, D. Rolnick, Complexity of linear regions in deep networks, in: ICML, 2019.

[69] B. Hanin, D. Rolnick, Deep ReLU networks have surprisingly few activation patterns, in: NeurIPS, 2019.

[70] M. Phuong, C. H. Lampert, Functional vs. parametric equivalence of ReLU networks, in: ICLR, 2020.

[71] D. Rolnick, K. Kording, Reverse-engineering deep ReLU networks, in: International Conference on Machine Learning (ICML), 2020.

[72] C. Hertrich, A. Basu, M. D. Summa, M. Skutella, Towards lower bounds on the depth of ReLU neural networks, in: NeurIPS, 2021.

[73] H. Tseran, G. Montúfar, On the expected complexity of maxout networks, in: NeurIPS, 2021.

[74] Y. Wang, Estimation and comparison of linear regions for ReLU networks, in: IJCAI, 2022.

[75] J. E. Grigsby, K. Lindsey, On transversality of bent hyperplane arrangements and the topological expressiveness of ReLU neural networks, SIAM Journal on Applied Algebra and Geometry (2022).

[76] G. Montúfar, Y. Ren, L. Zhang, Sharp bounds for the number of regions of maxout networks and vertices of Minkowski sums, SIAM Journal on Applied Algebra and Geometry (2022).

[77] C. A. Haase, C. Hertrich, G. Loho, Lower bounds on the depth of integral ReLU neural networks via lattice polytopes, in: ICLR, 2023.

[78] J. Huchette, G. Muñoz, T. Serra, C. Tsay, When deep learning meets polyhedral theory: A survey, arXiv:2305.00241 (2023).

[79] I. Safran, D. Reichman, P. Valiant, How many neurons does it take to approximate the maximum?, in: SODA, 2024.

[80] C. Hertrich, G. Loho, Neural networks and (virtual) extended formulations, arXiv:2411.03006 (2024).

[81] G. Averkov, C. Hojny, M. Merkert, On the expressiveness of rational ReLU neural networks with bounded depth, in: ICLR, 2025.

[82] J. Huchette, Advanced mixed-integer programming formulations: Methodology, computation, and application, Ph.D. thesis, Massachusetts Institute of Technology, 2018.

[83] A. M. Schweidtmann, A. Mitsos, Deterministic global optimization with artificial neural networks embedded, Journal of Optimization Theory and Applications (2019).

[84] C. Tsay, J. Kronqvist, A. Thebelt, R. Misener, Partition-based formulations for mixed-integer optimization of trained ReLU neural networks, in: NeurIPS, 2021.

[85] M. Fischetti, J. Jo, Deep neural networks and mixed integer linear optimization, Constraints (2018).

[86] B. Grimstad, H. Andersson, ReLU networks as surrogate models in mixed-integer linear programs, Computers & Chemical Engineering (2019).

[87] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, M. J. Kochenderfer, et al., Algorithms for verifying deep neural networks, Foundations and Trends® in Optimization (2021).

[88] F. Badilla, M. Goycoolea, G. Muñoz, T. Serra, Computational tradeoffs of optimization-based bound tightening in ReLU networks, arXiv:2312.16699 (2023).

[89] H. Zhao, H. Hijazi, H. Jones, J. Moore, M. Tanneau, P. V. Hentenryck, Bound tightening using rolling-horizon decomposition for neural network verification, in: CPAIOR, 2024.

[90] C. Hojny, S. Zhang, J. S. Campos, R. Misener, Verifying message-passing neural networks via topology-based bounds tightening, in: ICML, 2024.

[91] C. Plate, M. Hahn, A. Klimek, C. Ganzer, K. Sundmacher, S. Sager, An analysis of optimization problems involving ReLU neural networks, arXiv:2502.03016 (2025).

[92] V. Tjeng, K. Xiao, R. Tedrake, Evaluating robustness of neural networks with mixed integer programming, in: ICLR, 2019.

[93] K. Y. Xiao, V. Tjeng, N. M. Shafiullah, A. Madry, Training for faster adversarial robustness verification via inducing ReLU stability, in: ICLR, 2019.

[94] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, R. Misener, Efficient verification of (relu)-based neural networks via dependency analysis, in: AAAI, 2020.

[95] G. Perakis, A. Tsiourvas, Optimizing objective functions from trained ReLU neural networks via sampling, arXiv:2205.14189 (2022).

[96] J. Tong, J. Cai, T. Serra, Optimization over trained neural networks: Taking a relaxing walk, in: CPAIOR, 2024.

[97] M. Cacciola, A. Frangioni, A. Lodi, Structured pruning of neural networks for constraints learning, Operations Research Letters (2024).

[98] J. S. Bridle, Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition (1990).

[99] P. Bonami, A. Lodi, A. Tramontani, S. Wiese, On mathematical programming with indicator constraints, Mathematical Programming (2015).

[100] E. Frantar, D. Alistarh, SparseGPT: Massive language models can be accurately pruned in one-shot, in: ICML, 2023.

[101] X. Ma, G. Fang, X. Wang, LLM-Pruner: On the structural pruning of large language models, in: NeurIPS, 2023.

[102] M. Sun, Z. Liu, A. Bair, J. Z. Kolter, A simple and effective pruning approach for large language models, in: ICLR, 2024.

[103] M. Xia, T. Gao, Z. Zeng, D. Chen, Sheared LLaMA: Accelerating language model pre-training via structured pruning, in: ICLR, 2024.

[104] Y. Cheng, D. Wang, P. Zhou, T. Zhang, Model compression and acceleration for deep neural networks: The principles, progress, and challenges, IEEE Signal Processing Magazine (2018).

[105] H. Li, Z. Xu, G. Taylor, C. Studer, T. Goldstein, Visualizing the loss landscape of neural nets, in: NeurIPS, 2018.

[106] R. Sun, D. Li, S. Liang, T. Ding, R. Srikant, The global landscape of neural networks: An overview, IEEE Signal Processing Magazine 37 (2020).

[107] C. Riera, C. Rey, T. Serra, E. Puertas, O. Pujol, Training thinner and deeper neural networks: Jumpstart regularization, in: CPAIOR, 2022.

[108] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, N. Freitas, Predicting parameters in deep learning, in: NeurIPS, 2013.

[109] T. Jin, D. Roy, M. Carbin, J. Frankle, G. Dziugaite, On neural network pruning's effect on generalization, in: NeurIPS, 2022.

[110] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, A. Peste, Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, Journal of Machine Learning Research (2021).

[111] L. Liebenwein, C. Baykal, B. Carter, D. Gifford, D. Rus, Lost in pruning: The effects of pruning neural networks beyond test accuracy, in: MLSys, 2021.

[112] S. Hooker, A. Courville, G. Clark, Y. Dauphin, A. Frome, What do compressed deep neural networks forget?, arXiv:1911.05248 (2019).

[113] M. Paganini, Prune responsibly, arXiv:2009.09936 (2020).

[114] S. Hooker, N. Moorosi, G. Clark, S. Bengio, E. Denton, Characterising bias in compressed models, arXiv:2010.03058 (2020).

[115] C. Tran, F. Fioretto, J.-E. Kim, R. Naidu, Pruning has a disparate impact on model accuracy, in: NeurIPS, 2022.

[116] A. Good, J. Lin, H. Sieg, M. Ferguson, X. Yu, S. Zhe, J. Wieczorek, T. Serra, Recall distortion in neural network pruning and the undecayed pruning algorithm, in: NeurIPS, 2022.

[117] D. Blalock, J. Ortiz, J. Frankle, J. Guttag, What is the state of neural network pruning?, in: MLSys, 2020.

[118] S. Hanson, L. Pratt, Comparing biases for minimal network construction with back-propagation, in: NeurIPS, 1988.

[119] M. Mozer, P. Smolensky, Using relevance to reduce network size automatically, Connection Science (1989).

[120] S. Janowsky, Pruning versus clipping in neural networks, Physical Review A (1989).

[121] Y. LeCun, J. Denker, S. Solla, Optimal brain damage, in: NeurIPS, 1989.

[122] B. Hassibi, D. Stork, Second order derivatives for network pruning: Optimal Brain Surgeon, in: NeurIPS, 1992.

[123] B. Hassibi, D. Stork, G. Wolff, Optimal brain surgeon and general network pruning, in: IEEE International Conference on Neural Networks, 1993.

[124] G. Sourek, F. Zelezny, Lossless compression of structured convolutional models via lifting, in: ICLR, 2021.

[125] I. Ganev, R. Walters, Model compression via symmetries of the parameter space, 2022. URL: https://openreview.net/forum?id=8MN_GH4Ckp4.

[126] J. Frankle, M. Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks, in: ICLR, 2019.

[127] N. Lee, T. Ajanthan, P. Torr, SNIP: Single-shot network pruning based on connection sensitivity, in: ICLR, 2019.

[128] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: ICCV, 2017.

[129] J. Luo, J. Wu, W. Lin, ThiNet: A filter level pruning method for deep neural network compression, in: ICCV, 2017.

[130] A. Aghasi, A. Abdi, N. Nguyen, J. Romberg, Net-Trim: Convex pruning of deep neural networks with performance guarantee, in: NeurIPS, 2017.

[131] M. Ye, C. Gong, L. Nie, D. Zhou, A. Klivans, Q. Liu, Good subnetworks provably exist: Pruning via greedy forward selection, in: ICML, 2020.

[132] S. P. Singh, D. Alistarh, WoodFisher: Efficient second-order approximation for neural network compression, in: NeurIPS, 2020.

[133] S. Verma, J.-C. Pesquet, Sparsifying networks via subdifferential inclusion, in: ICML, 2021.

[134] X. Yu, T. Serra, S. Ramalingam, S. Zhe, The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks, in: ICML, 2022.

[135] R. Benbaki, W. Chen, X. Meng, H. Hazimeh, N. Ponomareva, Z. Zhao, R. Mazumder, Fast as CHITA: Neural network pruning with combinatorial optimization, in: ICML, 2023.

[136] A. Ebrahimi, D. Klabjan, Neuron-based pruning of deep neural networks with better generalization using kronecker factored curvature approximation, in: IJCNN, 2023.

[137] M. Cacciola, A. Frangioni, X. Li, A. Lodi, Deep neural networks pruning via the structured perspective regularization, SIAM Journal on Mathematics of Data Science (2023).

[138] D. Wu, I.-V. Modoranu, M. Safaryan, D. Kuznedelev, D. Alistarh, The iterative optimal brain surgeon: Faster sparse recovery by leveraging second-order information, in: NeurIPS, 2024.

[139] M. Turner, A. Chmiela, T. Koch, M. Winkler, SurrogateLIB: An extendable library of mixed- integer programs with embedded machine learning predictors, 2024. URL: https://doi.org/10.5281/zenodo.11231147.

[140] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE (1998).

[141] H. Xiao, K. Rasul, R. Vollgraf, Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, arXiv 1708.07747 (2017).