Structured Pruning of Neural Networks for Constraints Learning

Matteo Cacciola[a], Antonio Frangioni[b], Andrea Lodi[c]

matteo.cacciola@polymtl.ca
frangio@di.unipi.it
andrea.lodi@cornell.edu

[a]*CERC, Polytechnique Montréal, Montréal, Canada*
[b]*University of Pisa, Pisa, Italy*
[c]*Cornell Tech and Technion – IIT, New York, USA*

Abstract

In recent years, the integration of Machine Learning (ML) models with Operation Research (OR) tools has gained popularity across diverse applications, including cancer treatment, algorithmic configuration, and chemical process optimization. In this domain, the combination of ML and OR often relies on representing the ML model output using Mixed Integer Programming (MIP) formulations. Numerous studies in the literature have developed such formulations for many ML predictors, with a particular emphasis on Artificial Neural Networks (ANNs) due to their significant interest in many applications. However, ANNs frequently contain a large number of parameters, resulting in MIP formulations that are impractical to solve, thereby impeding scalability. In fact, the ML community has already introduced several techniques to reduce the parameter count of ANNs without compromising their performance, since the substantial size of modern ANNs presents challenges for ML applications as it significantly impacts computational efforts during training and necessitates significant memory resources for storage. In this paper, we showcase the effectiveness of pruning, one of these techniques, when applied to ANNs prior to their integration into MIPs. By pruning the ANN, we achieve significant improvements in the speed of the solution process. We discuss why pruning is more suitable in this context compared to other ML compression techniques, and we identify the most appropriate pruning strategies. To highlight the potential of this approach, we conduct experiments using feed-forward neural networks with multiple layers to construct adversarial examples. Our results demonstrate that pruning offers remarkable reductions in solution times without hindering the quality of the final decision, enabling the resolution of previously unsolvable instances.

Keywords: Artificial Neural Networks, Mixed Integer Programming, Model compression, Pruning

1. Introduction

The concept of embedding learned functions inside Mixed Integer Programming (MIP) formulations, also known as "Learning-Symbolic Programming" or "Constraint Learning", has gained attention in recent literature [1, 2, 3]. Furthermore, there has been an increase in the availability of tools that automatically embed commonly used predictive models into MIPs [4, 5, 6, 7]. These techniques and tools are especially valuable when employing ML models for predictions and utilizing OR methods for decision making based on those predictions. Unlike the traditional two-stage approaches [8], embedding the predictive model within the decision-making process in an end-to-end optimization framework has been shown to yield superior results. Examples of applications are automatic algorithmic configuration [9, 10], adversarial examples identification [11], cancer treatments development [12], and chemical process optimization [13, 14].

A very relevant case is when the learned function is an ANN, since ANNs are the state-of-the-art mod-

els for numerous essential ML tasks in Computer Vision and Natural Language processing. Consequently, there have been efforts in the literature to automate the embedding of ANNs [15]. For instance, [5] enables to incorporate feed-forward architectures with ReLU activation functions into MIPs, utilizing the output of the ANN in the objective function. The maturity of the field is demonstrated by the fact that one of the leading commercial MIP solvers, Gurobi, recently released a package that allows feed-forward ReLU networks to be part of MIP formulations, with compatibility for popular ML packages such as PyTorch, Keras, and scikit-learn.

Unfortunately, even when we consider simple architectures that have only ReLU activation functions, the representation of an ANN in a MIP will introduce binary variables, due to the combinatorial nature of the ReLU function. Additionally, the number of binary variables and the associated constraints that need to be added to the MIP is proportional to the number of parameters in the ANN. Deep Learning has witnessed a clear trend towards developing architectures with a very large number of parameters, which contributes to ANNs high predictive power and state-of-the-art performance in various applications. This, however, poses issues in terms of training costs, storage requirements, and prediction time. Consequently, numerous methods, known as model compression techniques, have been developed to reduce the size of ANNs without compromising their predictive capability. Yet, the large size of the ANNs presents an even more significant scalability challenge when it is embedded into a MIP, due to the potentially exponential growth of the latter computational cost with its size (and, in particular, the number of binary variables). Using a state-of-the-art network in a MIP formulation may easily result in an overwhelming number of binary variables and constraints, rendering the models unsolvable within a reasonable time using any available solver.

In this paper, we demonstrate that pruning methods, originally developed to address specific ML challenges, can be effectively applied in the context of embedding ANNs into MIPs. Specifically, we utilize a structured pruning technique that we previously developed to significantly accelerate the solution time for adversarial example identification problems using Gurobi.

The remainder of the paper is organized as follows: Section 2 provides a formal definition of the problem concerning the embedding of learned functions in MIP formulations. Additionally, it presents one of the existing formulations from the literature specifically designed for embedding ANNs. In Section 3, we introduce pruning techniques and we describe the specific pruning method employed in our experiments. Section 4 focuses on the benefits of pruning when incorporating ANNs into MIPs. We discuss the reasons why pruning is advantageous in this context and provide insights on selecting appropriate pruning techniques. Finally, in Section 5 we present numerical results to empirically validate that pruning can effectively speed up the solution process of MIPs with embedded ANNs.

2. Embedding learned functions in Mixed Integer Programs

We consider a general class of (Mixed-Integer) Nonlinear Programs with "learned constraints". That is, the formulation of the problem would need to involve some functions $g_i(x)$, $i = 1, \dots, k$, defined on the variable space of the optimization decisions, that are "hard" in the sense that no compact algebraic formulation, and not even an efficient computation oracle, is available. Yet, (large) data sets are available, or can be constructed, of outputs $\bar{y} = g_i(\bar{x})$ for given $\bar{x}$. These datasets can be used in several existing ML paradigms (Support Vector Machines, Decision Trees, ANNs, ...) to construct estimates $\bar{g}_i(x)$ of each $g_i(x)$, $i = 1, \dots, k$, with a workable algebraic description that can then be inserted into an optimization model. Thus, we consider the class of Mathematical Programs with Learned Constraints (MPLC)

$$\min\ cx + by \tag{1}$$
$$\text{s.t. } y_i = \bar{g}_i(x) \qquad i = 1, \dots, k \tag{2}$$
$$Ax + By \le d \tag{3}$$
$$x \in X \tag{4}$$

Linearity in (1) and (3) is not strictly necessary in our development, but it is often satisfied in applications (see, e.g., [5, 11, 12]) and we assume it for notational simplicity. Indeed, when X in (4) contains

integrality restrictions on (some of) the x variables, the class already contains Mixed-Integer Linear Programs (MILP), whose huge expressive power does not need to be discussed. Of course, a significant factor in the complexity of (1)–(4) is the algebraic form of the $\bar{g}_i(x)$, which impacts the class of optimization problems it ultimately belongs to. A significant amount of research is already available on formulations for embedding feedforward ANNs, in particular with ReLU activations, in a MIP context [11, 1, 2, 3, 16]. In these formulations, the neural network is constructed layer by layer. Denoting the input vector at layer ℓ as o_ℓ, and the corresponding weight matrix and bias vector as W_ℓ and b_ℓ, respectively, one has

$$o_{\ell+1} = \max(\, 0\,,\; W_\ell o_\ell + b_\ell\,)$$

that can be expressed in a MI(L)P form as

$$v_\ell^+ - v_\ell^- = W_\ell o_\ell + b_\ell \tag{5}$$
$$0 \le v_\ell^+ \le M^+ z_\ell \tag{6}$$
$$0 \le v_\ell^- \le M^-(1 - z_\ell) \tag{7}$$
$$o_{\ell+1} = v_\ell^+ \tag{8}$$
$$z_\ell \in \{\, 0\,,\; 1\,\}^m \tag{9}$$

Constraints (6) and (7) ensure that both v_ℓ^+ and v_ℓ^- are (component-wise) positive, and since the z_ℓ are (component-wise) binary, that at most one of them is positive. Consequently, constraint (5) forces the relations $v_\ell^+ = \max\{\, W_\ell o_\ell + b_\ell\,,\; 0\,\}$ and $v_\ell^- = \min\{\, W_\ell o_\ell + b_\ell\,,\; 0\,\}$ (of course, constraint (8) is only there to make apparent what the output of the layer is). Denoting by n the number of neurons in layer ℓ and by m the number of neurons in layer $\ell + 1$, system (5)–(9) contains m binary variables, $n + 2m$ continuous variables, and $3m$ constraints. A significant aspect of this model (fragment) is the use of big-M constraints (6) and (7). It is well known that the choice of the value for the constants M can significantly impact the time required to solve an instance. Indeed, the Optimized Big-M Bounds Tightening (OBBT) method has been developed in [11] to find effective values for this constant.

As previously mentioned, the state-of-the-art solver Gurobi now includes an open-source Python package that automatically embeds ANNs with ReLU activation into a Gurobi model. Additionally, starting from the 10.0.1 release, Gurobi has the capability to detect if a model contains a block of constraints representing the relationship $y = g(x)$, where $g(\cdot)$ is an ANN, in order to then apply the aforementioned OBBT techniques to enhance the solution process. Despite showing a substantial improvement with respect to the previous version, the capabilities of Gurobi to solve these MIPs are still limited. In particular, when embedding an ANN into a MIP, Gurobi is not able to solve the problem in a reasonable time unless the number of layers and neurons in the ANN is small.

3. Artificial Neural Networks Pruning

As mentioned in the introduction, the size of state-of-the-art ANNs has been growing exponentially over the years. While these models deliver remarkable performance, they come with high computational costs for training and inference, as well as substantial memory requirements for storage. To address this issue, various techniques have been developed to reduce these costs without significantly compromising the predictive power of the network. One such technique is pruning, which involves reducing the ANN size by eliminating unnecessary parameters. Consider for instance a linear layer with input x_{inp}, output x_{out}, and weight and bias tensors W and b, i.e., $x_{out} = W x_{inp} + b$. Thus, pruning entails removing certain entries from W or b. That is, pruning, say, the parameter $W_{1,1}$ results in the first coordinate of x_{inp} being ignored in the scalar product when computing the first coordinate of x_{out}.

Pruning individual weight entries can offer some advantages, but it is generally suboptimal. Since most of the computation is performed on GPUs, there is little computational benefit unless entire blocks of computation, such as tensor multiplications, are removed. Removing entire structures of the ANN is known as *structured* pruning, in contrast to *unstructured* pruning that involves eliminating single weights. In the example of the linear layer, structured pruning would aim to remove entire neurons by deleting rows from the W tensor (along with the corresponding b entry in most cases). Figures 1, 2, and 3 illustrate the difference between these two pruning

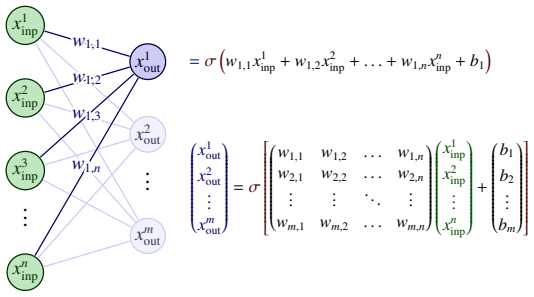

Figure 1: Unpruned network

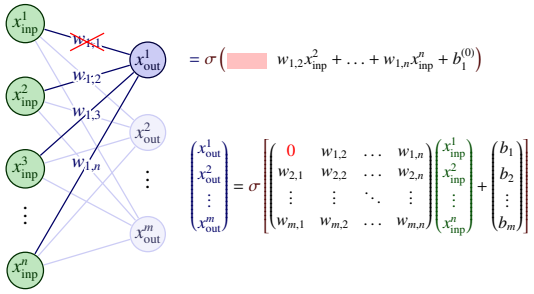

Figure 2: Unstructured pruning

techniques.

The literature on pruning techniques for neural networks is vast and encompasses a wide range of approaches. One simple and commonly used method is magnitude-based pruning, which involves removing parameters with small magnitudes. This was first introduced in [17] and has been widely adopted since. However, more sophisticated strategies have also been proposed, such as Bayesian methods [18, 19, 20, 21], combinations of pruning with other compression techniques [22, 23, 24], and zero accuracy drop pruning [25, 26, 27, 28].

A relevant subset of pruning techniques uses a regularization term to enforce sparsity in tensor weight. It is common practice in Machine Learning to add a regularization term $R(w)$ to the standard loss function $L(X, Y, w)$, where w is the vector containing the ANNs parameters and (X, Y) is the training set. Usually, $R(w)$ penalizes the magnitude of the parameters (e.g., $R(w) = |w|_2^2$) and it is known to improve the generalization performances of the model. If the form of $R(w)$ is chosen carefully, e.g., $R(w) = |w|_1$, it can also lead to a sparse parameter vector w. When a network parameter is zero, it can typically be removed without changing the model output for any given input. Hence, if $R(w)$ is chosen appropriately to induce all the weights of some neurons to be zero, then such neurons can be removed from the network. Many regularization terms have been proposed both for structured and unstructured pruning, including but not limited to l_1 norm, BerHu term [29], group lasso, and l_p/l_q norms [30].

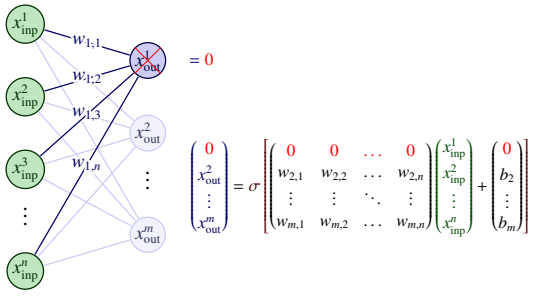

Figure 3: Structured pruning

3.1. The Structured Perspective Regularization Term

In the literature, the majority of pruning techniques rely on heuristics to determine the impact of removing a parameter or a structure from the ANN. This trend persists in recent works [31, 32, 33, 34], including methods that still utilize simple magnitude-based criteria [35, 36, 37, 38]. Only a few techniques attempt to develop a theoretically-grounded methodology [39, 40, 41, 18], and these methods do not primarily focus on structured pruning. In light of this, a pruning technique was developed in [42] that is motivated by strong theoretical foundations and specifically addresses structured pruning. In [42], the pruning problem is addressed by starting with a naïve exact MIP formulation and then deriving a stronger formulation by leveraging the Perspective Reformulation technique [43]. Analogously to what is done in [44, 45, 46] for individual variables rather than groups of them, an efficient way to solve the continuous relaxation of this problem is obtained by projecting away the binary variables, resulting in an equivalent problem to standard ANN training with the inclusion of the new *Structured Perspective Regularization* (SPR) term

$$z(W;\alpha,M) = \begin{cases} 2\sqrt{(1-\alpha)\alpha}||W||_2 & \text{if } \frac{||W||_\infty}{M} \le \sqrt{\frac{\alpha}{1-\alpha}}||W||_2 \le 1 \\ \frac{\alpha M}{||W||_\infty}||W||_2^2 + (1-\alpha)\frac{||W||_\infty}{M} & \text{if } \sqrt{\frac{\alpha}{1-\alpha}}||W||_2 \le \frac{||W||_\infty}{M} \le 1 \\ \alpha||W||_2^2 + (1-\alpha) & \text{otherwise,} \end{cases}$$

where M is a constant, α is a tunable hyperparameter and W is the weight tensor corresponding to the structure we want to prune (e.g., the weight matrix of a neuron). That is, in order to prune the ANN one trains it using as loss function

$$L(X, Y, W) + \lambda \sum_{j \in \mathcal{N}} z(W_j; \alpha, M),$$

where W_j is the weight matrix corresponding to neuron j and $\mathcal{N}$ is the set of neurons of the ANN. Coupled with a final magnitude-based pruning step, this approach has been shown to provide state-of-the-art pruning performances thanks to the unique and interesting properties of the SPR term. This potentially comes at the expense of extra hyperparameter tuning effort for α and M, which is unlikely to be a major issue in this application since ANNs that can be embedded in a MILP, even after pruning, cannot possibly have the extremely large size common in applications like Computer Vision and Naturale Language Processing, and therefore their training and tuning time is unlikely to be a major factor.

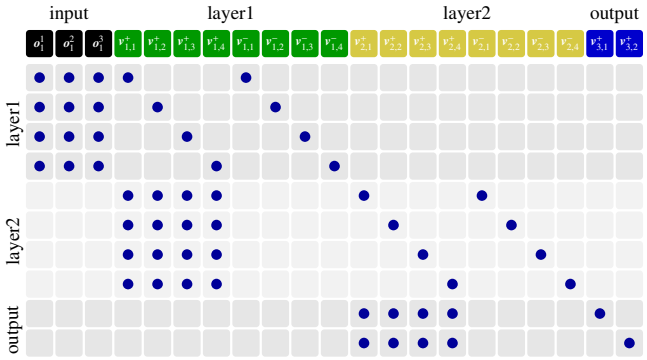

Figure 4: Constraints matrix

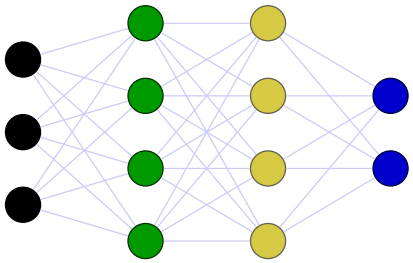

Figure 5: Corresponding network

4. Pruning as a Speed-Up Strategy

As previously mentioned, in the context of embedding ANNs in MIPs, scalability becomes a significant challenge as the number of (binary) variables and constraints grows proportionally with the number of parameters in the embedded ANN, but the cost of solving the MI(L)P may well grow exponentially in the number of (binary) variables. It therefore makes even more sense to employ the ML compression techniques that are used to reduce the computational resources required by ANNs. Many compression techniques other than pruning exist in the ML literature. However, not all of them are effective in the context of MIPs with embedded ANNs. For instance, quantization techniques aim to train networks that have weight values in a discrete (relatively small) set of $\mathbb{R}$ [47, 48]. One possibility is to directly implement the ANN using a lower bit number format than the standard Float-32 one [49, 50]. Quantization is a very popular technique in ML since can decrease both backward- and forward-pass computational effort, at the same time reducing the memory footprint of the resulting model. However, in the contest of MIPs, quantization does not bring any advantage, since the resulting problem from embedding a quantized ANN is not significantly different, from an Operations Research point of view, to the one where a non-quantized model has been embedded. Indeed, weights are coefficients in (5)–(9), and having them in a small set of (integer) values may at most have a minor impact on the solution time. Other methods, like low-rank decomposition and parameter-sharing techniques [51, 52, 53, 54], modify the internal operations of layers; this means that they cannot directly be used in this context without the development of new, specific formulations and new algorithms that can automatically detect them in a MIP problem.

By contrast, structured pruning techniques perfectly fit the needs of embedding an ANN in a MIP. Even unstructured pruning may have some impact, since when a weight is removed (i.e., set to zero) the corresponding entry in the MIP constraints matrix is also set to zero, leading to a sparser constraints matrix. However, entirely removing variables or constraints is more effective; in the case of a feed-forward ANN, this corresponds to performing structured pruning on neurons, as visualized in Figures 4- 5- 6- 7.

It is interesting to remark that, for ML purposes,

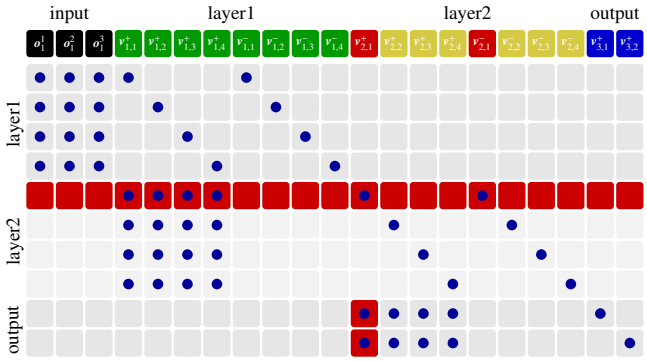

Figure 6: Constraints matrix, in red the removed constraints and variables due to neurons pruning

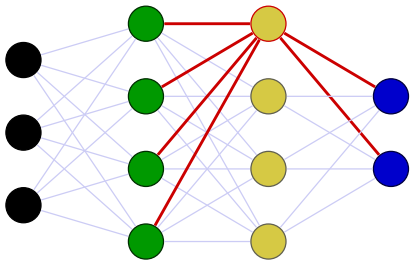

Figure 7: Corresponding network, the highlighted part is the pruned one.

pruning techniques only bring advantages at inference time, but reducing the number of parameters only reduces linearly the computational cost of the forward pass. By contrast, removing neurons of a network brings an exponential speed up in the time required to solve the resulting MIP formulations. Hence, pruning is arguably more relevant for OR than for ML, despite having been developed in the latter area. In particular, structured pruning—as opposed to unstructured one—is crucial in that it allows using existing automatic structure detection algorithms, such as that implemented in Gurobi, while unstructured pruning is very likely to result in a different structure of the constraints matrix that would not be recognizable, thereby preventing the use of OBBT techniques that are crucial in this context.

Based on the considerations above, we argue about modifying the existing pipeline for embedding ANNs in MIPs. After training the ANN (or during training, depending on the technique used), we prune the model before embedding it in the MIP formulation of the problem in hand. This approach either reduces the solution time of the MIP with the same generalisation performances, or, possibly, allows one to include larger, therefore more expressive ANNs, capable of achieving higher accuracy while still maintaining the ability to solve the resulting MIPs within a reasonable time. In particular, we will employ the Structured Perspective Regularization, i.e., we train the ANN by adding the SPR term to the loss, which will lead to a weight tensor with a structured sparsity. After fixing to zero (i.e., removing) neurons whose weights are all below a fixed threshold, we fine-tune the network with a standard loss for a few more epochs (see [42] for details). The obtained ANN is then embedded in the MIP, and it will require the addition of fewer variables and constraints with respect to its unpruned counterpart.

5. Experiments

5.1. Building adversarial examples

We test the effectiveness of pruning in the task of finding an adversarial example of a given network. In particular, we focus on the verification problem [55], which consists in finding a *slight* modification of an input that is originally correctly classified by the network in such a way that the modified one is assigned to a chosen class by the ANN. More formally, assume we are given a trained ANN $\bar{g}(\cdot) : \mathbb{R}^n \to [0,1]^C$ and one input x such that $\bar{g}(x)$ has its maximum value at the coordinate corresponding to the correct class of x. Denoting with k this coordinate and with h the coordinate with the second highest value of $\bar{g}(x)$, the problem we want to solve is

$$\max\ y_h - y_k \tag{10}$$
$$\text{s.t. } y = \bar{g}(\bar{x}) \tag{11}$$
$$\Delta \geq x - \bar{x} \tag{12}$$
$$\Delta \geq \bar{x} - x \tag{13}$$
$$\bar{x} \in \mathbb{R}^n \tag{14}$$

where Δ is a given distance bound. Clearly, (10)–(14) is a special case of the MPLC class (1)–(4). In particular, the constraint (11) encodes an ANN function, so it needs to be handled with the techniques we presented in Section 2. We selected this problem since it is of great interest to ML researchers. Furthermore, it can in principle be relevant to test the robustness of networks of any size, and therefore it

allows to explore the boundaries of what MPLC approaches (with or without pruning) can achieve.

5.2. General setup and notation

To test the effectiveness of our pruning techniques, we ran some experiments on network robustness using the MNIST dataset. We used the same settings of the notebook available at `https://github.com/Gurobi/gurobi-machinelearning/blob/main/notebooks/adversarial/adversarial_pytorch.ipynb`, where formulation (10)-(14) is solved with $\Delta = 5$.

We trained the ANNs using the Pytorch SGD optimizer with no weight decay and no momentum. We used 128 as batch size and we trained the network for 50 epochs with a constant learning rate equal to 0.1. All the networks are Pytorch sequential models containing only Linear and ReLU layers. For the pruned networks, we performed a (limited) 3 by 3 grid search to choose the λ factor that multiplies the SPR term and the α hyper-parameter needed in its definition (M is automatically set as in [42]). After 50 training epochs, the model is fine-tuned for 10 epochs without using any regularization. Note that the objective of the grid search is to find the smallest network that keeps basically the same out-of-sample accuracy of the original one, and better results could conceivably be obtained by employing end-to-end techniques that take into account the optimization process in the computation of the loss [56, 57].

In tables 1 and 2, the first column reports the network architecture of the used ANN and if pruning was used, while the Δ parameter value of (10)-(14) can be found in the first row. We compare the result of the baseline approach (i.e., without pruning) and the result obtained using the pruning method with the best hyper-parameters found. We report the validation accuracy (in percentage), the time needed by Gurobi to solve the obtained MIP (in seconds), and the number of branch-and-bound nodes explored during that time. Additionally, for the pruned networks, we report the value of λ and α and the architecture of the network after pruning. When referring to a network architecture, the terms LxN refer to a sequence of L layers each of them containing N neurons. When multiple terms follow each other, it indicates their order in the network. For example, 2x20-3x10 stands for a network that starts with 2 layers of 20 neurons and continues with 3 layers of 10 neurons. Each experiment is repeated 3 times and a time limit of 1800 seconds is given to Gurobi.

5.3. Detailed results

Table 1 shows the results using Δ=5 on 4 different architectures with an increasing number of neurons and layers. When pruning small architectures, like the 2x50 and 2x100 networks, pruning the ANN results in at least halving the time used by Gurobi. Moreover, the accuracy of the pruned models is higher than the baseline, this is, likely, because pruning has also a regularization effect.

The results on the 2x200 architecture show that the baseline is not able to solve the problems in the given time for two out of three runs. Instead, our method always leads to MIPs that are easily solved by Gurobi while maintaining the same accuracy as the baseline.

Finally, we report the results using the 6x100 networks, significantly bigger with respect to the previous ones. The baseline, once again, cannot solve two out of the three problems in the given time limit. Instead, our method is able to succeed in all cases, at the cost of losing a little bit of accuracy (0.3 percent in the best case).

As a last remark, we notice that for all the MIPs we solved relatively to unpruned network, no counterexample existed in the given neighborhood (i.e., the optimal value of (10)-(14) is negative). This remains true for the corresponding pruned counterparts, confirming that the pruned and unpruned versions of the MIPs are qualitatively very similar.

5.4. Investigating the quality of the solutions

To better validate the quality of our results, we solved again the adversarial problem (10)-(14) using Δ=20 and employing the same networks trained in the previous experiments. This was aimed to find adversarial examples in the given region to better understand the effect of pruning on the resulting MIP. We report the results in Table 2, where the "accuracy" and "pruned architecture" columns have been removed since they are the same as in the previous table. For all the experiments, a counter-example existed in the given region, and in the last column of Table 2, named "Found", we report if Gurobi was

Δ = 5					
Arch	λ-α	Acc.	Time	Nodes	Pruned Arch
2x50 Baseline		97.55	14.88	5820	
		97.47	20.65	12040	
		97.25	8.07	9497	
2x50 Pruned	0.5-0.9	97.77	3.29	3328	1x39-1x43
		97.49	3.93	6482	1x30-1x42
		97.73	1.96	3992	1x39-1x42
2x100 Baseline		97.96	39.29	2971	
		97.76	35.01	3112	
		97.97	39.00	3019	
2x100 Pruned	0.5-0.9	98.08	15.99	3066	1x61-1x80
		98.01	15.65	3107	1x61-1x87
		98.04	17.67	2951	1x63-1x86
2x200 Baseline		98.14	1800.37	424758	
		98.04	1800.18	401361	
		97.95	781.90	58656	
2x200 Pruned	0.5-0.5	97.96	18.66	3029	1x56-1x144
		98.13	24.65	3600	1x57-1x144
		98.04	28.19	2997	1x59-1x140
6x100 Baseline		97.60	474,76	15261	
		97.77	1800.02	798306	
		97.67	818.19	14334	
6x100 Pruned	1.0-0.1	97.52	231.02	3173	1x39-1x82 2x61-1x60-1x54
		97.47	79.22	7566	1x44-1x71-1x49 -1x53-1x49-1x45
		97.21	44.24	11417	1x37-1x72-1x48 -1x51-1x48-1x46

Table 1: Results using Δ = 5.

able to find one adversarial example in the given time limit. Unsurprisingly, for all the MIPs corresponding to pruned networks, Gurobi was able to find an adversarial example within a time considerably inferior to the 1800 seconds limit. Moreover, all the adversarial examples obtained using a pruned network were also adversarial for the unpruned counterpart with the same starting architecture. This empirically proved that, in our setting, pruning can be even used to solve the adversarial example problem for the unpruned counterpart and it is again a good indication that pruning does not heavily affect the resulting MIP. This is in accordance with the ML literature, where there is a good consensus that not-too-aggressive pruning of ANNs does not significantly impacts their robustness [58, 59], and therefore the existence—or not—of the counter-example in our application. Finally, the times reported in Table 2 show that the speed-up is still very significant even with the new value of Δ and that in some cases the Baseline is not able to find any adversarial example.

We conclude this section by noting that additional experiments, which are not included in this paper for the sake of brevity, have shown that a high setting of the OBBT parameter [11] of Gurobi is crucial to obtain good performances both for pruned and unpruned instances, confirming the importance of structured pruning.

Δ = 20				
Arch	λ-α	Time	Nodes	Found
2x50 Baseline		1.85	1	YES
		5.06	1221	YES
		1.66	1	YES
2x50 Pruned	0.5-0.9	2.73	127	YES
		2.90	1128	YES
		0.64	1	YES
2x100 Baseline		17.35	1217	YES
		147.67	7532	YES
		102.67	3252	YES
2x100 Pruned	0.5-0.9	6.66	2079	YES
		6.03	127	YES
		2.16	1	YES
2x200 Baseline		439.17	40075	YES
		563.24	17597	YES
		508.14	6014	YES
2x200 Pruned	0.5-0.5	2.56	1	YES
		18.65	5433	YES
		9.60	1202	YES
6x100 Baseline		1800.06	138918	NO
		1800.03	237328	NO
		1800.10	184954	NO
6x100 Pruned	1.0-0.1	15.53	1	YES
		7.51	1	YES
		129.70	27045	YES

Table 2: Results using Δ = 20.

6. Conclusions and future directions

This paper has demonstrated the effectiveness of pruning artificial neural networks in accelerating the solution time of mixed-integer programming problems that incorporate ANNs. The choice of the sparsity structure for pruning plays a crucial role in achieving significant speed-up, and we argued that structured pruning is superior to unstructured one. Further research in this area can focus on gaining a deeper understanding of which sparsity structures are most suitable for improving the solution time of MIPs. Exploring the trade-off between pruning-induced sparsity and solution quality is another interesting avenue for future investigations. By advancing our understanding of pruning techniques and

their impact on MIPs, we can enhance the efficiency and scalability of embedding ANNs in optimization problems.

Acknowledgments

The authors are grateful to Pierre Bonami for his generous and insightful feedback. This work has been supported by the NSERC Alliance grant 544900- 19 in collaboration with Huawei-Canada

References

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

[2] D. Bienstock, G. Muñoz, S. Pokutta, Principled deep neural network training through linear programming, arXiv preprint arXiv:1810.03218 (2018).

[3] R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, J. P. Vielma, Strong mixed-integer programming formulations for trained neural networks, Mathematical Programming 183 (1-2) (2020) 3–39.

[4] A. Thebelt, J. Kronqvist, M. Mistry, R. M. Lee, N. Sudermann-Merx, R. Misener, Entmoot: A framework for optimization over ensemble tree models, Computers & Chemical Engineering 151 (2021) 107343. doi:https://doi.org/10.1016/j.compchemeng.2021.107343.
URL https://www.sciencedirect.com/science/article/pii/S0098135421001216

[5] D. Bergman, T. Huang, P. Brooks, A. Lodi, A. U. Raghunathan, Janos: An integrated predictive and prescriptive modeling framework, INFORMS Journal on Computing 34 (2) (2022) 807–816. arXiv:https://doi.org/10.1287/ijoc.2020.1023, doi:10.1287/ijoc.2020.1023.
URL https://doi.org/10.1287/ijoc.2020.1023

[6] D. Maragno, H. Wiberg, D. Bertsimas, S. I. Birbil, D. d. Hertog, A. Fajemisin, Mixed-integer optimization with constraint learning, arXiv preprint arXiv:2111.04469 (2021).

[7] F. Ceccon, J. Jalving, J. Haddad, A. Thebelt, C. Tsay, C. D. Laird, R. Misener, Omlt: Optimization & machine learning toolkit, The Journal of Machine Learning Research 23 (1) (2022) 15829–15836.

[8] K. J. Ferreira, B. H. A. Lee, D. Simchi-Levi, Analytics for an online retailer: Demand forecasting and price optimization, Manufacturing & service operations management 18 (1) (2016) 69–88.

[9] G. Iommazzo, C. D'Ambrosio, A. Frangioni, L. Liberti, A learning-based mathematical programming formulation for the automatic configuration of optimization solvers, in: Machine Learning, Optimization, and Data Science: 6th International Conference, LOD 2020, Siena, Italy, July 19–23, 2020, Revised Selected Papers, Part I 6, Springer, 2020, pp. 700–712.

[10] G. Iommazzo, C. d'Ambrosio, A. Frangioni, L. Liberti, Learning to configure mathematical programming solvers by mathematical programming, in: Learning and Intelligent Optimization: 14th International Conference, LION 14, Athens, Greece, May 24–28, 2020, Revised Selected Papers 14, Springer, 2020, pp. 377–389.

[11] M. Fischetti, J. Jo, Deep neural networks and mixed integer linear optimization, Constraints 23 (3) (2018) 296–309.

[12] D. Bertsimas, A. O'Hair, S. Relyea, J. Silberholz, An analytics approach to designing combination chemotherapy regimens for cancer, Management Science 62 (5) (2016) 1511–1531.

[13] I. Fahmi, S. Cremaschi, Process synthesis of biodiesel production plant using artificial neural networks as the surrogate models, Computers & Chemical Engineering 46 (2012) 105–123.

[14] F. A. Fernandes, Optimization of fischer-tropsch synthesis using neural networks, Chemical Engineering & Technology: Industrial Chemistry-Plant Equipment-Process Engineering-Biotechnology 29 (4) (2006) 449–453.

[15] B. Grimstad, H. Andersson, Relu networks as surrogate models in mixed-integer linear programs, Computers & Chemical Engineering 131 (2019) 106580.

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

[17] S. Han, B. Dally, Efficient methods and hardware for deep learning, University Lecture (2017).

[18] D. Molchanov, D. Vetrov, A. Ashukha, Variational dropout sparsifies deep neural networks, in: 34th International Conference on Machine Learning, ICML 2017, 2017, pp. 3854–3863.

[19] G. Bellec, D. Kappel, W. Maass, R. Legenstein, Deep rewiring: Training very sparse deep networks, in: International Conference on Learning Representations, 2018.
URL https://openreview.net/forum?id=BJ_wN01C-

[20] C. Louizos, K. Ullrich, M. Welling, Bayesian compression for deep learning, Advances in neural information processing systems 30 (2017).

[21] Y. Zhou, Y. Zhang, Y. Wang, Q. Tian, Accelerate cnn via recursive bayesian pruning, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 3306–3315.

[22] J. M. Alvarez, M. Salzmann, Compression-aware training of deep networks, Advances in neural information processing systems 30 (2017).

[23] Y. Ma, R. Chen, W. Li, F. Shang, W. Yu, M. Cho, B. Yu, A unified approximation framework for compressing and accelerating deep neural networks, in: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, 2019, pp. 376–383.

[24] E. Frantar, D. Alistarh, Optimal brain compression: A framework for accurate post-training quantization and pruning, in: A. H. Oh, A. Agarwal, D. Belgrave, K. Cho (Eds.), Advances in Neural Information Processing Systems, 2022.
URL https://openreview.net/forum?id=ksVGCOlOEba

[25] C. Baykal, L. Liebenwein, I. Gilitschenski, D. Feldman, D. Rus, Sensitivity-informed provable pruning of neural networks, SIAM Journal on Mathematics of Data Science 4 (1) (2022) 26–45.

[26] M. E. Halabi, S. Srinivas, S. Lacoste-Julien, Data-efficient structured pruning via submodular optimization, in: A. H. Oh, A. Agarwal, D. Belgrave, K. Cho (Eds.), Advances in Neural Information Processing Systems, 2022.
URL https://openreview.net/forum?id=K2QGzyLwpYG

[27] J. Chee, M. Renz, A. Damle, C. D. Sa, Model preserving compression for neural networks, in: A. H. Oh, A. Agarwal, D. Belgrave, K. Cho (Eds.), Advances in Neural Information Processing Systems, 2022.
URL https://openreview.net/forum?id=gt-l9Hu2ndd

[28] X. Yu, T. Serra, S. Ramalingam, S. Zhe, The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks, in: K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, S. Sabato (Eds.), Proceedings of the 39th International Conference on Machine Learning, Vol. 162 of Proceedings of Machine Learning Research, PMLR, 2022, pp. 25668–25683.
URL https://proceedings.mlr.press/v162/yu22f.html

[29] S. Lambert-Lacroix, L. Zwald, The adaptive berhu penalty in robust regression, Journal of Nonparametric Statistics 28 (3) (2016) 487–514.

[30] T. Chen, B. Ji, T. Ding, B. Fang, G. Wang, Z. Zhu, L. Liang, Y. Shi, S. Yi, X. Tu, Only train once: A one-shot neural network training and pruning framework, Advances in Neural Information Processing Systems 34 (2021) 19637–19651.

[31] M. Shen, H. Yin, P. Molchanov, L. Mao, J. Liu, J. M. Alvarez, Structural pruning via latency-saliency knapsack, in: A. H. Oh, A. Agarwal, D. Belgrave, K. Cho (Eds.), Advances in Neural Information Processing Systems, 2022.
URL https://openreview.net/forum?id=cUOR-_VsavA

[32] S. Yu, A. Mazaheri, A. Jannesari, Topology-aware network pruning using multi-stage graph embedding and reinforcement learning, in: International Conference on Machine Learning, PMLR, 2022, pp. 25656–25667.

[33] J. Rachwan, D. Zügner, B. Charpentier, S. Geisler, M. Ayle, S. Günnemann, Winning the lottery ahead of time: Efficient early network pruning, in: International Conference on Machine Learning, PMLR, 2022, pp. 18293–18309.

[34] E. Frantar, D. Alistarh, SPDY: Accurate pruning with speedup guarantees, in: K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, S. Sabato (Eds.), Proceedings of the 39th International Conference on Machine Learning, Vol. 162 of Proceedings of Machine Learning Research, PMLR, 2022, pp. 6726–6743.
URL https://proceedings.mlr.press/v162/frantar22a.html

[35] Y. Li, S. Gu, K. Zhang, L. V. Gool, R. Timofte, Dhp: Differentiable meta pruning via hypernetworks, in: European Conference on Computer Vision, Springer, 2020, pp. 608–624.

[36] T.-W. Chin, R. Ding, C. Zhang, D. Marculescu, Towards efficient model compression via learned global ranking, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 1518–1528.

[37] Y. Zhang, Y. Yao, P. Ram, P. Zhao, T. Chen, M. Hong, Y. Wang, S. Liu, Advancing model pruning via bi-level optimization, in: A. H. Oh, A. Agarwal, D. Belgrave, K. Cho (Eds.), Advances in Neural Information Processing Systems, 2022.
URL https://openreview.net/forum?id=t6O08FxvtBY

[38] N. Gamboa, K. Kudrolli, A. Dhoot, A. Pedram, Campfire: Compressible, regularization-free, structured sparse training for hardware accelerators, arXiv preprint arXiv:2001.03253 (2020).

[39] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, Y. Wang, A systematic dnn weight pruning framework using alternating direction method of multipliers, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 184–199.

[40] M. A. Carreira-Perpinan, Y. Idelbayev, "learning-compression" algorithms for neural net pruning, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 8532–8541. doi:10.1109/CVPR.2018.00890.

[41] C. Louizos, M. Welling, D. P. Kingma, Learning sparse neural networks through l_0 regularization, in: International Conference on Learning Representations, 2018.
URL https://openreview.net/forum?id=H1Y8hhg0b

[42] M. Cacciola, A. Frangioni, X. Li, A. Lodi, Deep neural networks pruning via the structured perspective regularization (2022). arXiv:2206.14056.
URL https://arxiv.org/abs/2206.14056

[43] C. D'Ambrosio, A. Frangioni, C. Gentile, Strengthening the sequential convex minlp technique by perspective reformulations, Optimization letter 13, Springer 13 (2019).
URL https://doi.org/10.1007/s11590-018-1360-9

[44] A. Frangioni, C. Gentile, E. Grande, A. Pacifici, Projected perspective reformulations with applications in de-

sign problems, Operations Research 59 (5) (2011) 1225–1232.

[45] A. Frangioni, F. Furini, C. Gentile, Approximated perspective relaxations: a project&lift approach, Computational Optimization and Applications 63 (3) (2016) 705–735.

[46] A. Frangioni, F. Furini, C. Gentile, Improving the approximated projected perspective reformulation by dual information, Operations Research Letters 45 (5) (2017) 519–524.

[47] M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: Training deep neural networks with binary weights during propagations, Advances in neural information processing systems 28 (2015).

[48] H. Wu, P. Judd, X. Zhang, M. Isaev, P. Micikevicius, Integer quantization for deep learning inference: Principles and empirical evaluation, arXiv preprint arXiv:2004.09602 (2020).

[49] M. Cacciola, A. Frangioni, M. Asgharian, A. Ghaffari, V. P. Nia, On the convergence of stochastic gradient descent in low-precision number formats, arXiv preprint arXiv:2301.01651 (2023).

[50] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 2704–2713.

[51] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, Z. Xu, Learning compact recurrent neural networks with block-term tensor decomposition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[52] M. Jaderberg, A. Vedaldi, A. Zisserman, Speeding up convolutional neural networks with low rank expansions, CoRR abs/1405.3866 (2014). arXiv:1405.3866.
URL http://arxiv.org/abs/1405.3866

[53] D. Eigen, J. Rolfe, R. Fergus, Y. LeCun, Understanding deep architectures using a recursive convolutional network, CoRR (12 2013).
URL https://arxiv.org/abs/1312.1847

[54] R. Dabre, A. Fujita, Recurrent stacking of layers for compact neural machine translation models, CoRR abs/1807.05353 (2018). arXiv:1807.05353.
URL http://arxiv.org/abs/1807.05353

[55] B. Batten, P. Kouvaros, A. Lomuscio, Y. Zheng, Efficient neural network verification via layer-based semidefinite relaxations and linear cuts., in: IJCAI, 2021, pp. 2184–2190.

[56] J. Mandi, E. Demirović, P. J. Stuckey, T. Guns, Smart predict–and–optimize for hard combinatorial optimization problems, in: Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI Press, 2020, pp. 1603–1610.

[57] J. Mandi, V. Bucarey, M. M. K. Tchomba, T. Guns, Decision-focused learning: through the lens of learning to rank, in: International Conference on Machine Learning, PMLR, 2022, pp. 14935–14947.

[58] X. Zhuang, Y. Ge, B. Zheng, Q. Wang, Adversarial network pruning by filter robustness estimation, in: ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2023, pp. 1–5. doi:10.1109/ICASSP49357.2023.10095291.

[59] M. Ayle, B. Charpentier, J. Rachwan, D. Zügner, S. Geisler, S. Günnemann, On the robustness and anomaly detection of sparse neural networks, arXiv preprint arXiv:2207.04227 (2022).