OAMIP: Optimizing ANN Architectures Using Mixed-Integer Programming

Mostafa ElAraby[1,3(✉)], Guy Wolf[1,4], and Margarida Carvalho[2,3]

[1] Mila – Quebec AI institute, Montreal, Canada
moustafa.elarabi@Umontrea.ca
[2] CIRRELT, Montreal, Canada
[3] Department of Computer Science and Operations Research, Université de Montréal, Montreal, Canada
[4] Department of Mathematics and Statistics, Université de Montréal, Montreal, QC, Canada

Abstract. In this work, we concentrate on the problem of finding a set of neurons in a trained neural network whose pruning leads to a marginal loss in accuracy. To this end, we introduce *Optimizing ANN Architectures using Mixed-Integer Programming* (OAMIP) to identify critical neurons and prune non-critical ones. The proposed OAMIP uses a Mixed-Integer Program (MIP) to assign importance scores to each neuron in deep neural network architectures. The impact of simultaneous neuron pruning on the main learning tasks guides the neurons' scores. By carefully devising the objective function of the MIP, we drive the solver to minimize the number of critical neurons (i.e., with high importance score) that maintain the overall accuracy of the trained neural network. Our formulation identifies optimized sub-network architectures that generalize across different datasets, a phenomenon known as lottery ticket optimization. This optimized architecture not only performs well on a single dataset but also generalizes across multiple ones upon retraining of network weights. Additionally, we present a scalable implementation of our pruning methodology by decoupling the importance scores across layers using auxiliary networks. Finally, we validate our approach experimentally, showing its ability to generalize on different datasets and architectures.

Keywords: Pruning Neural Networks · Mixed Integer Programming · Neurons Ranking · Sparse Neural Networks

1 Introduction

Deep learning has proven its power to solve complex tasks and to achieve state-of-the-art results in various domains such as image classification, speech recognition, machine translation, robotics and control [6,24]. Over-parameterized artificial neural networks (ANN), which have more parameters than the training

G. Wolf and M. Carvalho—Equal contribution.

A. A. Cire (Ed.): CPAIOR 2023, LNCS 13884, pp. 219–237, 2023.
https://doi.org/10.1007/978-3-031-33271-5_15

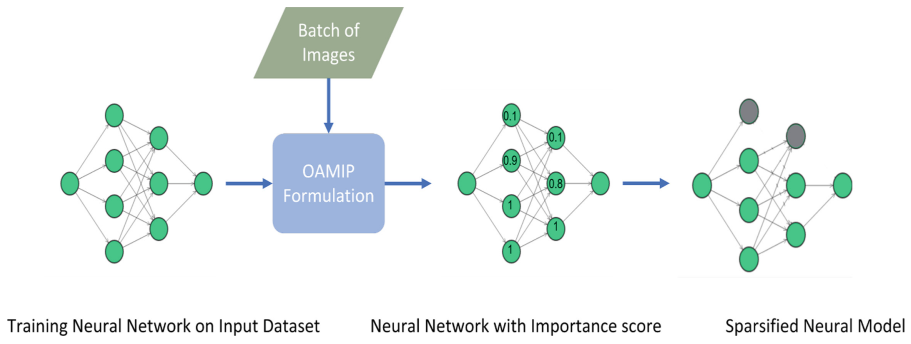
Batch of Images
OAMIP Formulation
0.1
0.9
1
1
0.1
0.8
1
Training Neural Network on Input Dataset
Neural Network with Importance score
Sparsified Neural Model

Fig. 1. The generic flow of OAMIP used to remove neurons with an importance score below a specific threshold.

samples, can be used to achieve state-of-the-art results in various tasks [39,57]. However, the large number of parameters comes at the expense of computational cost in terms of memory footprint, training time, and inference time on resource-limited devices.

In this context, the pruning of neurons in an over-parameterized neural model has been an active area of research, enabling the increase of computational efficiency and the uncovering of sub-networks with marginal (or even no) loss in the network's predictive capacity [1,9,17,28,41,42,45,50,51,56]. The typical sparsification procedure involves training a neural model to convergence, computing the parameters' importance, then pruning existing ones using specific criteria, and fine-tuning the neural model to regain its lost accuracy. Existing pruning and neuron ranking procedures [1,9,17,18,27,35,45,56] require iterations of fine-tuning on the sparsified model instead of pruning a pre-trained network directly. Moreover, the evaluation of the generalization of sparsified models across different datasets is under-explored in existing pruning and neuron ranking procedures [31], which is consistent with the lottery ticket hypothesis [13,34,37].

We remark that modern network architectures often use sparse neuron connectivity and, most notably, convolutional layers in image processing. Indeed, the limited size of the parameter space in such cases increases the effectiveness of network training and enables the learning of meaningful semantic features from the input images [15]. Inspired by the benefits of sparsity in such architecture designs, we aim to leverage the neuron sparsity achieved by our framework, Optimizing ANN Architectures using Mixed-Integer Programming (OAMIP) to obtain optimized neural architectures that can generalize well across different datasets. For this purpose, we create a sparse sub-network by optimizing on one dataset and then training the same architecture, i.e., masked, on another dataset. Our results indicate a promising direction of future research into the utilization of combinatorial optimization for effective automatic architecture tuning to augment handcrafted network architecture design.

Contributions and Paper Organization. In OAMIP, illustrated in Fig. 1, we formalize the notation of *neuron importance score* in a trained neural network and the associated dataset. The neuron importance score reflects how much activity decrease can be inflicted while controlling the loss on the neural network model accuracy. To this end, in Sect. 2, we begin by providing background on the constraints that serve as the basis for our Mixed-Integer Programming (MIP) formulation presented in Sect. 3. Concretely, we propose a MIP that allows the computation of the importance score for each fully connected neuron and convolutional feature map. The error propagation associated with pruning between different layers defines each neuron's importance score. In addition, we also discuss the extension of the MIP constraints for other layers besides ReLU-activated fully connected layers. Section 4 describes OAMIP in detail, namely the integration of the neuron importance scores on the pruning procedure. Here, we also propose a methodology to independently decouple the computation of neuron importance score per layer to represent deeper architectures and, thus, scale up our approach to models like VGG-16 [44]. Furthermore, in Sect. 5, we show OAMIP's robustness to various input data points besides its ability to parallelize the computation of importance score per class. Finally, we show that OAMIP's importance score generalizes well over various datasets complying with the lottery ticket hypothesis [13].

1.1 Related Work

Weight Pruning Methods. Early methods in weight pruning relied on the weight magnitude by disabling the lowest magnitude weights and re-training/fine-tuning the resulting sub-network [16,29,37]. Magnitude-based techniques rely on the intuition that large weight values are more critical during inference than smaller weight values. [36] devised a greedy criteria-based pruning with fine-tuning by back-propagation. The criteria devised are given by the absolute difference between dense and sparse neural model loss (ranker) to avoid a drop in the predictive capacity. [43] developed a framework that computes the neurons' importance at each layer through a single backward pass as an approximation to the interpretability of each neuron during inference. Other related techniques, using different objectives and interpretations of neuron importance, have been presented [1,3,19,20,22,54], and require either fine-tuning to recover the network's performance or dynamic re-training and pruning. Another line of research [10,33,42,49, 53,55] formulates an optimization model to select which neuron to disable without losing performance on the task at hand. With a less conservative perspective but also using an optimization-based model, OAMIP aims to quantify a generalizable per-neuron importance score for either a pre-trained network or at initialization without re-training or fine-tuning the network. Similarly, other pruning procedures aim to avoid the fine-tuning step by pruning the network during initialization. In particular, SNIP [28] and GraSP [51] focus on predicting critical weights during initialization via salience scores and then train the sub-network until convergence. SNIP [28] was the first to investigate the pruning of a network during initialization by computing the connection's sensitivity to an input batch of data through gradient back-propagation. OAMIP can be applied to the network at initialization or after training without requiring a long fine-tuning step.

Lottery Ticket. [13] introduced the lottery ticket theory that shows the existence of a lucky pruned sub-network, a *winning ticket*. The lucky pruned sub-network can be trained effectively with fewer parameters while achieving a marginal loss in accuracy. [37] proposed "one ticket to win them all" for sparsifying n over-parameterized trained neural models based on the lottery hypothesis. Searching for the winning ticket involves pruning the model and disabling some of its sub-networks. The pruned model can be trained on a different dataset using the same initialization (winning ticket), achieving good results. To this end, the dataset used for the pruning phase must be sufficiently large. The lucky sub-network is found by iteratively pruning the lowest magnitude weights and re-training. Another phenomenon discovered in [40,52] was the existence of smaller, high-accuracy models that reside in larger random networks. This phenomenon is called the strong lottery ticket hypothesis, which was proven [34] on ReLU fully connected layers. Furthermore, [51] proposed a technique to select the winning ticket at initialization (before training the ANN) by computing an importance score based on the gradient flow in each unit.

Mixed-Integer Programming. [12] presented a Mixed-Integer Linear Programming big-M formulation to represent trained ReLU neural networks. Later, [4] introduced the strongest possible tightening to the big-M formulation by adding strengthening separation constraints when needed, which reduced the solving time by several orders of magnitude. Recently, [48] presented efficient partitioning strategies that improved solving time. All the proposed formulations are designed to represent trained ReLU ANNs with fixed parameters. In our framework, we use the formulation from [12] since its performance was good due to our tight local variable bounds, and its polynomial number of constraints (while the models in [4,48] are non-compact). The interest of representing an ANN as a MIP lies in its use to evaluate robustness, carry out compression and create adversarial examples for trained ANNs. For instance, [21,47] used a big-M formulation to evaluate the robustness of neural models against adversarial attacks. [55] modeled an extension of the optimal brain surgeon [18], where the goal is to select and remove the weights that have the most negligible impact on the predictive capacity of the network as an Integer Quadratic Program. However, the optimal brain surgeon pruning criteria rely heavily on the weights scale. Moreover, the weights' scale will be sensitive to the architecture used; different normalization layers affect the scale and magnitude of weights in a different way [28]. [42] also used a MIP formulation to maximize the compression of a trained neural network without decreasing predictive accuracy. *Lossless compression* [42] relies on different compression methods, such as removing neurons and folding layers. However, the reported computational experiments lead only to the removal of inactive neurons. OAMIP can identify such neurons and quantify the importance of various neurons with respect to the predictive capacity while pruning neurons that are non-critical across different datasets. The latter means that the sub-networks found by our framework to a specific dataset generalize to others.

2 Preliminaries

Consider layer l of a trained ReLU neural network with $\boldsymbol{W^l}$ as the weight matrix, w_i^l as row i of $\boldsymbol{W^l}$, and b^l as the bias vector. For each input data point x, let h^l be a decision vector denoting the output value of layer l, i.e., $h^l = ReLU(\boldsymbol{W^l} h^{l-1} + b^l)$ for $l > 0$ and $h^0 = x$, and z_i^l be a binary variable taking value 1 if the unit i is active ($w_i^l h^{l-1} + b_i^l \geq 0$) and 0 otherwise. Finally, let L_i^l and U_i^l be constants indicating a valid lower and upper bound for the input of each neuron i in layer l. We discuss the computation of these bounds in Sect. 3.2. For now, we assume that L_i^l and U_i^l are sufficiently small and large numbers, respectively, i.e., the so-called big-M values. Next, we provide the representation of ReLU neural networks of [12]. Although [4] proposed an ideal MIP formulation with an exponential number of facet-defining constraints that can be separated efficiently, we use the formulation by [12], since it performed well in practice for our purpose. For the sake of simplicity, we describe the formulation for one layer l of the model at neuron i and one input data point x:

$$h_i^0 = x_i \tag{1a}$$

$$h_i^l \geq 0, \quad \text{for } l > 0 \tag{1b}$$

$$h_i^l + (1 - z_i^l) L_i^l \leq w_i^l h^{l-1} + b_i^l, \tag{1c}$$

$$h_i^l \leq z_i^l U_i^l, \tag{1d}$$

$$h_i^l \geq w_i^l h^{l-1} + b_i^l, \tag{1e}$$

$$z_i^l \in \{0, 1\}, h_i^l \in \mathbb{R}. \tag{1f}$$

In constraint (1a), the initial decision vector h^0 is forced to be equal to the input x of the first layer. When z_i^l is 0, constraints (1b) and (1d) force h_i^l to be zero, reflecting a non-active neuron. If an entry of z_i^l is 1, then constraints (1c) and (1e) enforce h_i^l to be equal to $w_i^l h^{l-1} + b_i^l$. After formulating the ReLU, if we relax the binary constraint (1f) on z_i^l to $[0, 1]$, we obtain a polyhedron, over which it is easier and faster to optimize. The *quality* (tightness) of such relaxation highly depends on the choice of tight upper and lower bounds, U_i^l, L_i^l. Indeed, the determination of tight bounds reduces the search space and hence, the solving time.

3 Neuron Importance Score

In what follows, we adapt constraints (1) to quantify neurons' importance, we describe the computation of the bounds L_i^l and U_i^l and we discuss the objective function for our MIP. Our goal is to compute importance scores for all layers in the model in an integrated fashion. In fact, [54] has shown that this integrated perspective leads to better predictive accuracy than layer by layer.

3.1 MIP Constraints

In ReLU-activated layers, we keep the previously introduced binary variables z_i^l and continuous variables h_i^l. Recall that these variables are linked to an input data point x, so if more than one data point is considered, copies of these variables must be created. Additionally, we create the continuous decision variables $s_i^l \in [0,1]$ representing neuron i importance score in layer l; contrarily to z_i^l and h_i^l, no copies of s_i^l are created for each input data point. In this way, we proceed to modify the ReLU constraints (1) by adding the neuron importance decision variable s_i^l to constraints (1c) and (1e):

$$h_i^l + (1 - z_i^l)L_i^l \leq w_i^l h^{l-1} + b_i^l - (1 - s_i^l)\max(U_i^l, 0), \tag{2a}$$

$$h_i^l \geq w_i^l h^{l-1} + b_i^l - (1 - s_i^l)\max(U_i^l, 0). \tag{2b}$$

Constraints (2) impose that when neuron i is activated due to the input h^{l-1}, i.e., $z_i^l = 1$, then h_i^l is equal to the right-hand-side of those constraints. This value can be directly decreased by reducing the neuron importance s_i^l. When neuron i is non-active, i.e., $z_i^l = 0$, constraint (2b) becomes irrelevant as its right-hand-side is negative. This fact together with constraints (1b) and (1d), imply that h_i^l is zero. Now, we claim that constraint (2a) allows s_i^l to be zero if that neuron is indeed non-important, i.e., for all possible input data points, neuron i is not activated. This claim can be shown through the following observations. Note that decisions h and z must be replicated for each input data point x as they represent the propagation of x over the neural network. On the other hand, s evaluates the importance of each neuron for the main learning task, and thus, it must be the same for all data input points. Thus, the key ingredients are the bounds L_i^l and U_i^l that are computed for each input data point, as explained in Sect. 3.2. In this way, if U_i^l is non-positive, s_i^l can be zero without interfering with constraints (2). The latter is driven by the objective function derived in Sect. 3.3. We designate a neuron as *critical* with respect to a trained ANN, if its importance score is higher than a predefined threshold, otherwise it is called *non-critical*.

We now discuss other architectures. Concerning convolutional feature maps, we convert them to toeplitz matrices and their input images to vectors. This allows us to use simple matrix multiplication which is computationally efficient and generates the full convolution output. For padded convolution we use only parts of the output of the full convolution, and for strided convolutions we use sum of 1 strided convolution as proposed by [7]. Moreover, we can represent the convolutional layer using the same formulation of fully connected layers presented in (2a). The importance score of convolutional layers is associated with each feature map [30,36].

We represent both max and average (avg) pooling on multi-input units in our MIP formulation. Pooling layers are used to reduce spatial representation of input images by applying an arithmetic operation on each feature map of the previous layer. Avg pooling layers compute the average operation on each feature map of the previous layer l having N^l as the number of neurons. This operation is linear and thus, it can directly be included in the MIP constraints:

$$h^{l+1} = \text{AvgPool}(h_1^l, \cdots, h_{N^l}^l) = \frac{1}{N^l}\sum_{i=1}^{N^l} h_i^l.$$

Max Pooling takes the maximum of each feature map of the previous layer:

$$h^{l+1} = \text{MaxPool}(h_1^l, \cdots, h_{N^l}^l) = \max\{h_1^l, \cdots, h_{N^l}^l\}.$$

This operation can be expressed by introducing a set of binary variables $m_1, \cdots, m_{N^l}$, where $m_i = 1$ implies $x = \text{MaxPool}(h_1^l, \cdots, h_{N^l}^l)$:

$$\sum_{i=1}^{N^l} m_i = 1$$

$$\left.\begin{aligned} x &\geq h_i^l, \\ x &\leq h_i^l m_i + U_i(1 - m_i) \\ m_i &\in \{0,1\} \end{aligned}\right\} i = 1, \cdots, N^l.$$

3.2 Bound Propagation

In the previous section, we assumed a large upper bound U_i^l and a small lower bound L_i^l. However, using large bounds may lead to long computational times and a loss of freedom to reduce the importance score, as discussed above. In order to overcome these issues, we tailor these bounds accordingly with their respective input point x by considering small perturbations on its value:

$$L^0 = x - \epsilon \tag{3a}$$

$$U^0 = x + \epsilon \tag{3b}$$

$$L^l = \boldsymbol{W}^{(l-)} U^{l-1} + \boldsymbol{W}^{(l+)} L^{l-1} \tag{3c}$$

$$U^l = \boldsymbol{W}^{(l+)} U^{l-1} + \boldsymbol{W}^{(l-)} L^{l-1} \tag{3d}$$

$$\boldsymbol{W}^{(l-)} \triangleq \min(\boldsymbol{W}^{(l)}, 0) \tag{3e}$$

$$\boldsymbol{W}^{(l+)} \triangleq \max(\boldsymbol{W}^{(l)}, 0). \tag{3f}$$

Propagating the initial bounds of the input data points throughout the trained model will create the desired bound using a simple arithmetic interval. The obtained bounds are tight, narrowing the space of feasible solutions.

3.3 MIP Objective

Our framework aims at identifying non-critical neurons without significantly decreasing the predictive accuracy of the pruned ANN. To this end, we combine two optimization objectives.

Our first objective is to maximize the set of neurons sparsified from the trained ANN. Recall that N^l is the number of neurons at layer l, and let n be the number of layers, and $I^l = \sum_{i=1}^{N^l}(s_i^l - 2)$ be the sum of neuron importance scores at layer l with s_i^l scaled down to the range $[-2, -1]$.

In order to create a relation between neurons' importance score in different layers, our objective becomes the maximization of the number of neurons sparsified from the $n-1$ layers with higher score I^l. Hence, we denote $A = \{I^l : l = 1, \ldots, n\}$ and formulate the sparsity loss as

$$\text{sparsity} = \frac{\max\limits_{A' \subset A, |A'|=(n-1)} \sum\limits_{I \in A'} I}{\sum_{l=1}^{n} |N^l|}. \tag{4}$$

Here, the goal is to maximize the number of non-critical neurons at each layer relative to the other layers of the trained neural model. Note that only the $n-1$ layers with the most significant importance score will weigh in the objective, allowing to reduce the pruning effort on some layers that will naturally have low scores. The total number of neurons then normalizes the sparsity quantification.

Our second objective is to minimize the loss of *important* information due to the sparsification of the trained neural model. Additionally, we aim for this minimization to be done without relying on the values of the logits, which are closely correlated with the neurons pruned at each layer. Otherwise, this would drive the MIP to simply give a total score of 1 to all neurons to keep the same output logit value. Instead, we formulate this optimization objective using the marginal softmax proposed in [14]. Using marginal softmax allows the solver to focus on minimizing the misclassification error without relying on logit values. Moreover, the scale of logits can be marginally different between the decision vector h^n computed by the MIP with some disabled neurons and the trained neural network predictions. To that end, in the proposed marginal softmax loss, the label with the highest logit value is optimized regardless of its value. Formally, we write the objective

$$\text{softmax} = \sum_{i=1}^{N^n} \log\left[\sum_c \exp(h_{i,c}^n)\right] - \sum_{i=1}^{N^n}\sum_c Y_{i,c} h_{i,c}^n, \tag{5}$$

where the index c stands for the class label. The softmax marginal objective retains the trained model's correct predictions for the batch of input images x having a one-hot encoded label Y without regard to the logit value. Finally, we combine the two objectives to formulate the loss

$$\text{loss} = \text{sparsity} + \lambda \cdot \text{softmax} \tag{6}$$

as a weighted sum of sparsification regularizer and marginal softmax.

4 OAMIP: Pruning Approach

Given a trained neural network and a dataset, our goal is to identify and prune non-critical neurons based on importance score s_i^l for neuron i at layer l. To this end, we formulated a neural network as a mixed-integer program, including the neuron importance score in its constraints and objective function. Algorithm 1 summarizes the integration of our formulation within a pruning procedure.

Algorithm 1: OAMIP: Optimizing ANN Architectures using a MIP

```
Require: Trained ANN, dataset $D$ and a threshold.
Ensure: Sub-network selected from the trained ANN.
1: Select a per-class image $D' \subset D$ to be fed into the MIP.
2: Solve the MIP restricted to $D'$ and save the neurons importance scores $s$.
3: Remove every neuron $i$ from layer $l$ with $s_i^l \leq$ threshold from the ANN.
4: Return pruned ANN (sub-network).
```

[58] highlights the phenomenon of neural collapse, where features of images from the same distribution in the training set collapse around a class mean and are maximally distant between different classes. Moreover, the neurons that are important for a specific class, as computed on an image, should not change drastically when another image from the same distribution as the training set is used. Besides, using all the training samples as input to the MIP solver is intractable. Hence, we use only a subset of the data points, each representing a class in the classification task for which we aim to approximate the neuron importance score (step 1). Then, OAMIP computes an estimation of the importance score of each neuron (step 2). With a small tuned threshold based on the network's architecture, we mask (prune) non-critical neurons with a score lower than the threshold (step 3). Finally, our proposed framework returns a pruned ANN (sub-network), achieving marginal loss in accuracy.

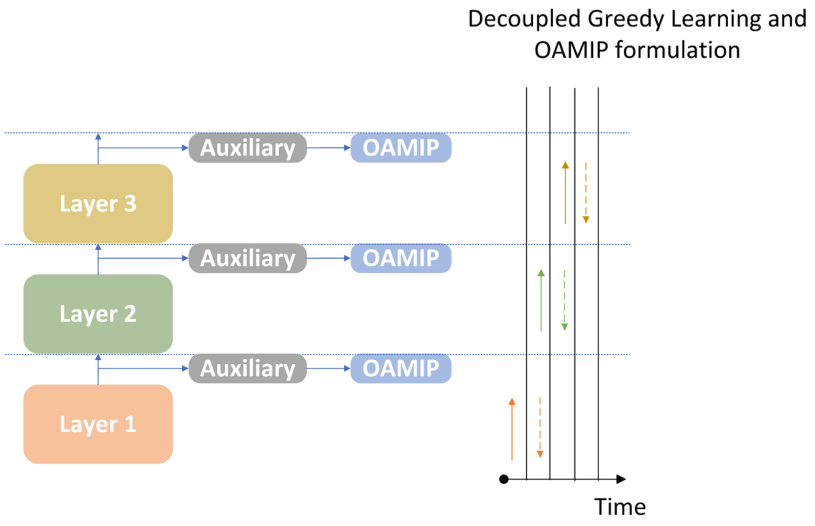

Fig. 2. Illustration of the auxiliary network attached to each sub-module along with the signal backpropagation during training as shown in [5].

The most time-sensitive step of OAMIP is the optimization of the MIP. The number of variables and constraints increases with the number of neurons and input data points. Indeed, if large and realistic ANNs are modeled with our MIP, the computation time for determining importance scores is expected to become very large, as observed in the problem tackled in [12]. To overcome the computational time issue, we propose independent computation of importance scores per layer using auxiliary networks [5]. In particular, we used decoupled greedy learning [5] to train each layer of VGG-16 [44] using a small auxiliary network, and, in this way, we computed the neuron importance score independently on each auxiliary network, as shown in Fig. 2. Then, we fine-tuned the generated masks for one epoch to propagate the errors across them resulting from the independent optimization. Decoupled training of each layer allowed us to represent deep models using the MIP formulation and to parallelize the computation per layer.

5 Empirical Results

This section shows experimentally that *(i)* our approach can efficiently find high-performance sub-networks from ANN architectures, *(ii)* the computed sub-networks generalize well to new datasets, and *(iii)* OAMIP outperforms the state-of-the-art approach SNIP with regards to generalization.

Experimental Setting. We used a simple fully connected 3-layer ANN (FC-3) model, with 300+100 hidden units, from [26], and another simple fully connected 4-layer ANN (FC-4) model, with 200+100+100 hidden units. In addition, we used the convolutional LeNet-5 [26] consisting of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully-connected layers. The largest architecture investigated was VGG-16 [44] consisting of a stack of convolutional (Conv.) layers with a small receptive field: 3×3. The VGG-16 was adapted for CIFAR-10 [25], having two fully connected layers of size 512 and average pooling instead of max pooling. Each of these models was trained three times with different initialization.

All models were trained for 30 epochs using RMSprop [46] optimizer with 1e-3 learning rate for MNIST and Fashion MNIST. LeNet-5 [26] on CIFAR-10 was trained using the SGD optimizer with learning rate 1e−2 and 256 epochs. VGG-16 [44] on CIFAR-10 was trained using Adam [23] with 1e−2 learning rate for 30 epochs. The hyper-parameters were tuned on the validation set's accuracy. All images were resized to 32 by 32 and converted to 3 channels to generalize the pruned network across different datasets. Our experiments revealed that $\lambda = 5$ generally provides the right trade-off between our two objectives (6) based on the validation set results; see the following thesis [11] for details on these experiments.

Computational Environment. The experiments were performed in an Intel(R) Xeon(R) CPU @ 2.30 GHz with 12 GB RAM and Tesla k80 using Mosek 9.1.11 [38] solver on top of CVXPY [2,8] and PyTorch 1.3.1[1].

[1] The code can be found here: https://github.com/chair-dsgt/mip-for-ann.

5.1 OAMIP Robustness

We examine the robustness of OAMIP against different batches of input images fed into the MIP, on the implementation of step 2 of OAMIP. Namely, we used 25 randomly sampled balanced images from the validation set. Figure 3 shows that changing the input images used by the MIP to compute neuron importance scores in step 2 resulted in marginal changes in the test accuracy between different batches. We remark that the input batches may contain images that were misclassified by the neural network. In this case, the MIP tries to use the score s to obtain the true label, which explains the variations in the pruning percentage. Furthermore, we show empirically that OAMIP is robust on different convergence levels of the trained neural network as shown in Fig. 4. Hence, we do not need to wait for the ANN to be trained to identify the target sub-network (strong lottery ticket hypothesis theory [34]).

Additionally, we experiment parallelizing per class neuron importance score computation using a balanced and imbalanced set of images per class. For those experiments, we sampled a random number of images per class (IMIDP), then

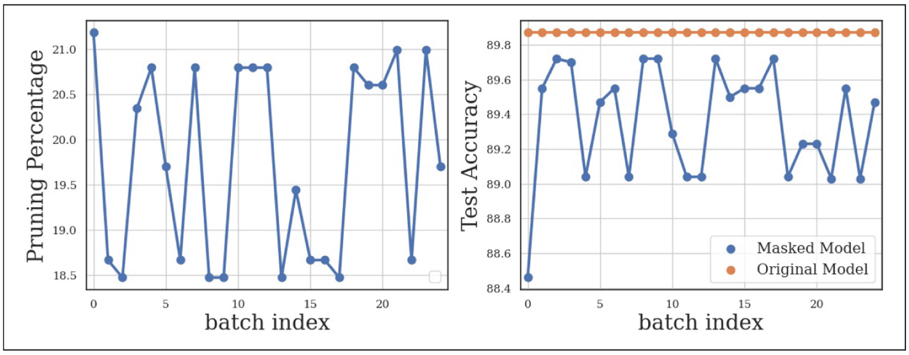

Fig. 3. Effect of changing validation set of input images.

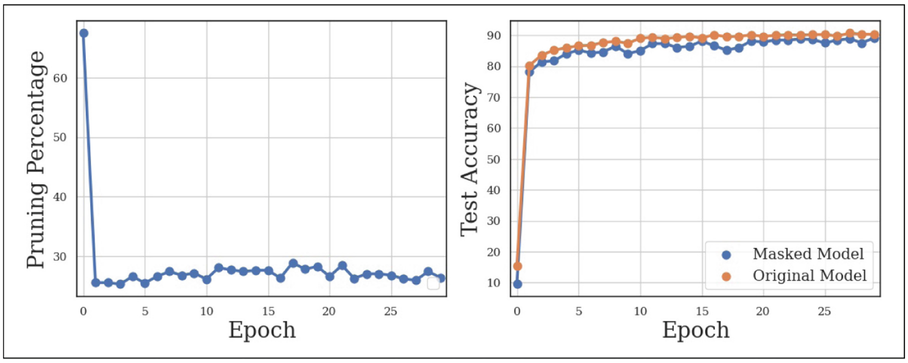

Fig. 4. Evolution of the computed masked sub-network during model training.

we took the average of the computed neuron importance scores from solving the MIP on each class. The obtained sub-networks were compared to solving the MIP with 1 image per class (IDP) and to solving the MIP with balanced images representing all classes (SIM). We achieved comparable results in terms of test accuracy and pruning percentage.

Table 1. Comparing test accuracy of Lenet-5 on imbalanced independent class by class (IMIDP.), balanced independent (IDP.) and simultaneously all classes (SIM) with 0.01 threshold, and $\lambda = 1$.

	MNIST	Fashion-MNIST
Ref	$98.8\% \pm 0.09$	$89.5\% \pm 0.3$
IDP.	$98.6\% \pm 0.15$	$87.3\% \pm 0.3$
Prune (%)	$19.8\% \pm 0.18$	$21.8\% \pm 0.5$
IMIDP.	$98.6\% \pm 0.1$	$88\% \pm 0.1$
Prune (%)	$15\% \pm 0.1$	$18.1\% \pm 0.3$
SIM.	$98.4\% \pm 0.3$	$87.9\% \pm 0.1$
Prune (%)	$13.2\% \pm 0.42$	$18.8\% \pm 1.3$

To conclude on the robustness of the scores computed based on the input points used in the MIP, we empirically show in Table 1 that our method is scalable, and that class contribution can be decoupled without deteriorating the approximation of neuron scores and thus, the performance of our methodology. Moreover, we show that OAMIP is robust even when an imbalanced number of data points per class (IMIDP) is used in the MIP formulation.

5.2 Comparison to Random and Critical Pruning

We started by training a reference model (REF.) using previously described training parameters. After training and evaluating the reference model on the test set, we fed an input batch of images from the validation set to the MIP. Then, the MIP solver computed the neuron importance scores based on those input images. We used 10 images in our experimental setup, each representing a class.

To validate our pruning policy guided by the computed importance scores, we created different sub-networks of the reference model, where the same number of neurons is removed in each layer, thus allowing a fair comparison among them. These sub-networks were obtained through different procedures: non-critical (our methodology), critical, and randomly pruned neurons. For VGG-16 experiments, an extra fine-tuning step for 1 epoch is performed on all generated sub-networks. Although we pruned the same number of neurons, which accordingly with [32] should result in similar performances, Table 2 shows that pruning non-critical neurons results in marginal loss and gives better performance. On the other hand, we observe a significant drop in the test accuracy when critical or a random set of neurons are removed compared with the reference model. If we fine-tune for just 1 epoch the sub-network obtained through our method, the model's accuracy can surpass the reference model. This is due to the fact that the MIP while computing neuron scores, is solving its marginal softmax (5) on true labels.

Table 2. Pruning results on fully connected (FC-3, FC-4) and convolutional (Lenet-5, VGG-16) network architectures using three different datasets. We compare the test accuracy between the unpruned reference network (REF.), randomly pruned model (RP.), model pruned based on critical neurons selected by the MIP (CP.) and our non-critical pruning approach with (OAMIP + FT) and without (OAMIP) fine-tuning for 1 epoch.

		Ref.	RP.	CP.	OAMIP	OAMIP+FT	Prune (%)	Runtime (s)
MNIST	FC-3	98.1%	83.6%	44.5%	**95.9%**	**97.8%**	44.5%	12 s
		±0.1	±4.6	±7.2	±0.87	±0.2	±7.2	±0.7
	FC-4	97.9%	77.1%	50%	**96.6%**	**97.6%**	42.9%	9s
		±0.1	±4.8	±15.8	±0.4	±0.01	±4.5	±0.4
	LeNet-5	98.9%	56.9%	38.6%	**98.7%**	**98.9%**	17.2%	1 s
		±0.1	±36.2	±40.8	±0.1	±0.04	±2.4	±0.6
Fashion-MNIST	FC-3	87.7%	35.3%	11.7%	**80%**	**88.1%**	68%	16 s
		±0.6	±6.9	±1.2	±2.7	±0.2	±1.4	±1
	FC-4	88.9%	38.3%	16.6%	**86.9%**	**88%**	60.8%	10 s
		±0.1	±4.7	±4.1	±0.7	±0.03	±3.2	±0.8
	LeNet-5	89.7%	33%	28.6%	**87.7%**	**89.8%**	17.8%	10 s
		±0.2	±24.3	±26.3	±2.2	±0.4	±2.1	±1
CIFAR-10	LeNet-5	72.2%	50.1%	27.5%	**67.7%**	**68.6%**	9.9%	6 s
		±0.2	±5.6	±1.7	±2.2	±1.4	±1.4	±0.5
	VGG-16	83.9%	85%	83.3%	N/A[a]	**85.3%**	36%	N/A[b]
		±0.4	±0.4	±0.3		±0.2	±1.1	

[a] A fine-tuning step is required to connect the results of independent layers.
[b] Computation was applied independently on each layer.

5.3 Generalization Between Different Datasets

Table 3. Cross-dataset generalization: sub-network masking is computed on source dataset (d_1) and then applied to target dataset (d_2) by re-training with the same early initialization. Test accuracies are presented for masked and unmasked (REF.) networks on d_2, as well as pruning percentage.

Model	Source dataset d_1	Target dataset d_2	REF. Acc.	Masked Acc.	Pruning (%)
LeNet-5	Mnist	Fashion MNIST	89.7% ± 0.3	89.2% ± 0.5	16.2% ± 0.2
		CIFAR-10	72.2% ± 0.2	68.1% ± 2.5	
VGG-16	CIFAR-10	MNIST	99.1% ± 0.1	99.4% ± 0.1	36% ± 1.1
		Fashion-Mnist	92.3% ± 0.4	92.1% ± 0.6	

In this experiment, we train the model on a dataset d_1 and create a masked neural model using our approach. After creating the masked model, we restart it to its original initialization. Finally, the new masked model is re-trained on another dataset d_2, and its generalization is analyzed.

Table 3 displays our experiments and respective results. When we compare generalization results to pruning using our approach on Fashion-MNIST and CIFAR-10, we discover that computing the critical sub-network for the LeNet-5 architecture on MNIST creates a more sparse sub-network. Moreover, this sub-network has a test accuracy better than zero-shot pruning without fine-tuning and comparable accuracy with the original ANN. This behavior occurs because the solver is optimizing on a batch of images that are classified correctly with high confidence from the trained model. Furthermore, computing the critical VGG-16 sub-network architecture on CIFAR-10 using decoupled greedy learning [5] generalizes well to Fashion-MNIST and MNIST.

5.4 Comparison to SNIP

OAMIP can be viewed as a compression technique of over-parameterized neural models. We compare it to SNIP [28].

SNIP computes connection sensitivities in a data-dependent way before the training. The sensitivity of a connection represents its importance based on the influence of the connection on the loss function. After computing the sensitivity, the connections below a predefined threshold are pruned before training (single shot).

In our methodology, we exclusively identify the importance of neurons and essentially prune all the connections of non-important ones. On the other hand, SNIP only focuses on pruning individual connections. Moreover, we highlight that SNIP can only compute connection sensitivity on the initialization of an ANN. Indeed, for a trained ANN, the magnitude of the derivatives concerning the loss function optimized during the training, makes SNIP keener to keep all the parameters. On the other hand, OAMIP can work on different convergence levels, as shown in Sect. 3.3. Furthermore, the connection sensitivity computed by SNIP is only network and dataset-specific; thus, the computed connection sensitivity for a single connection does not give a meaningful signal about its general importance for a given task. Rather, it needs to be compared to the sensitivity of other connections.

In order to bridge the differences between the two methods and provide a fair comparison in equivalent settings, we make a slight adjustment to our method. In step 2 of OAMIP, we compute neuron importance scores on the model's initialization[2]. We note that we used only 10 images as input to the MIP, corresponding to the 10 different classes, and 128 images as input to SNIP, following its original paper [28]. Our algorithm was able to prune neurons from fully connected and convolutional layers of LeNet-5. After creating the sparse

[2] Remark: we used $\lambda = 1$ and pruning threshold 0.2 and kept ratio 0.45 for SNIP. Training procedures as in Sect. 5.

networks using SNIP and our methodology, we trained them on the Fashion-MNIST dataset. The difference between SNIP (88.8% ± 0.6) and our approach (88.7% ± 0.5) was marginal in terms of test accuracy. SNIP pruned 55% of the ANN's parameters and OAMIP 58.4%.

Table 4. Cross-dataset generalization comparison between SNIP, with neurons having the lowest sum of connections' sensitivity pruned, and our framework (OAMIP), both applied on initialization, see Sect. 5.3 for the generalization experiment description.

Source dataset d_1	Target dataset d_2	REF. Acc.	Method	Masked Acc.	Pruning (%)
Mnist	Fashion-MNIST	89.7% ± 0.3	SNIP	85.8% ± 1.1	53.5% ± 1.8
			OAMIP	**88.5% ± 0.3**	**59.1% ± 0.8**
	CIFAR-10	72.2% ± 0.2	SNIP	53.5% ± 3.3	53.5% ± 1.8
			OAMIP	**63.6% ± 1.4**	**59.1% ± 0.8**

Next, we compare SNIP and OAMIP in terms of generalization. In Table 4, we show that our framework outperforms SNIP in terms of generalization. We adjusted SNIP to prune entire neurons based on the value of the sum of its connections' sensitivity, and our framework was also applied to ANN's initialization. When our framework is applied on the initialization, more neurons are pruned as the marginal softmax part of the objective function discussed in Sect. 3.3 is weighing less ($\lambda = 1$), driving the optimization to focus on model sparsification.

Finally, we remark that the adjustments made to SNIP and OAMIP in the previous experiments are solely for comparison, while (unlike SNIP) the primary purpose of our method is to allow optimization at any stage – before, during, or after training. In the specific case of optimizing at initialization and discarding entire neurons based on aggregated connection sensitivity, the SNIP approach may have some advantages, notably in scalability for deep architectures. However, it also has some limitations, as previously discussed.

6 Discussion

We proposed a mixed integer program to compute neuron importance scores in ReLU-based deep neural networks. Our contributions focus on providing scalable computations of importance scores in fully connected and convolutional layers. We presented results showing that these scores can effectively prune unimportant parts of the network without significantly affecting its predictive capacity. Further, our results indicate that this approach allows the automatic construction of efficient sub-networks that can be transferred and retrained on different datasets. Knowing a neural network's critical components can further impact future work beyond the pruning applications presented here.

Acknowledgements. This work was partially funded by: IVADO (l'institut de valorisation des données) [*G. W.*, *M.C.*]; FRQ-IVADO Research Chair in Data Science for Combinatorial Game Theory, and NSERC grant 2019-04557 [*M.C.*] Canada CIFAR AI Chair, NIH grant R01GM135929 [*G. W.*].

References

1. Adamczewski, K., Park, M.: Dirichlet pruning for neural network compression. Proc. Mach. Learn. Res. **130** (2021)
2. Agrawal, A., Verschueren, R., Diamond, S., Boyd, S.: A rewriting system for convex optimization problems. J. Control Decis. **5**(1), 42–60 (2018)
3. Amjad, R.A., Liu, K., Geiger, B.C.: Understanding neural networks and individual neuron importance via information-ordered cumulative ablation. IEEE Trans. Neural Netw. Learn. Syst. **33**, 7842–7852 (2021)
4. Anderson, R., Huchette, J., Tjandraatmadja, C., Vielma, J.P.: Strong mixed-integer programming formulations for trained neural networks. In: International Conference on Integer Programming and Combinatorial Optimization, pp. 27–42. Springer (2019)
5. Belilovsky, E., Eickenberg, M., Oyallon, E.: Decoupled greedy learning of CNNs. In: Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 736–745. PMLR (2020)
6. Bengio, Y., Goodfellow, I., Courville, A.: Deep Learning, vol. 1. Citeseer (2017)
7. Brosch, T., Tam, R.: Efficient training of convolutional deep belief networks in the frequency domain for application to high-resolution 2D and 3D images. Neural Comput. **27**(1), 211–227 (2015)
8. Diamond, S., Boyd, S.: CVXPY: a Python-embedded modeling language for convex optimization. J. Mach. Learn. Res. **17**(83), 1–5 (2016)
9. Dong, X., Chen, S., Pan, S.: Learning to prune deep neural networks via layer-wise optimal brain surgeon. In: Advances in Neural Information Processing Systems, pp. 4857–4867 (2017)
10. Ebrahimi, A., Klabjan, D.: Neuron-based pruning of deep neural networks with better generalization using kronecker factored curvature approximation. arXiv preprint arXiv:2111.08577 (2021)
11. ElAraby, M.: Optimizing ANN architectures using mixed-integer programming. Master's dissertation, Université de Montréal (2020). http://hdl.handle.net/1866/24312
12. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints **23**(3), 296–309 (2018)
13. Frankle, J., Carbin, M.: The lottery ticket hypothesis: finding sparse, trainable neural networks. In: International Conference on Learning Representations (2019)
14. Gimpel, K., Smith, N.A.: Softmax-margin CRFs: training log-linear models with cost functions. In: The Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 733–736. Association for Computational Linguistics (2010)
15. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
16. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv preprint arXiv:1510.00149 (2015a)
17. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, pp. 1135–1143 (2015)
18. Hassibi, B., Stork, D.G., Wolff, G.J.: Optimal brain surgeon and general network pruning. In: IEEE International Conference on Neural Networks, pp. 293–299. IEEE (1993)

19. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI 2018, pp. 2234–2240. AAAI Press (2018)
20. Hooker, S., Erhan, D., Kindermans, P.J., Kim, B.: A benchmark for interpretability methods in deep neural networks. In: Advances in Neural Information Processing Systems, pp. 9734–9745 (2019)
21. Huang, P.S., et al.: Achieving verified robustness to symbol substitutions via interval bound propagation. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 4083–4093 (2019)
22. Jordao, A., Yamada, F., Schwartz, W.R.: Deep network compression based on partial least squares. Neurocomputing **406**, 234–243 (2020)
23. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. In: Proceedings of the 3rd International Conference for Learning Representations (ICLR 2015), San Diego (2015)
24. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
25. Krizhevsky, A.: Learning multiple layers of features from tiny images. Master's thesis, University of Toronto (2009)
26. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
27. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems, pp. 598–605 (1990)
28. Lee, N., Ajanthan, T., Torr, P.H.S.: SNIP: single-shot network pruning based on connection sensitivity. In: International Conference on Learning Representations (ICLR) (2019)
29. Lei, W., Chen, H., Wu, Y.: Compressing deep convolutional networks using k-means based on weights distribution. In: Proceedings of the 2nd International Conference on Intelligent Information Processing, pp. 1–6 (2017)
30. Li, Y., Adamczewski, K., Li, W., Gu, S., Timofte, R., Van Gool, L.: Revisiting random channel pruning for neural network compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 191–201 (2022)
31. Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X.: Pruning and quantization for deep neural network acceleration: a survey. Neurocomputing **461**, 370–403 (2021)
32. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. In: International Conference on Learning Representations (2018)
33. Luo, J.H., Wu, J., Lin, W.: ThiNet: a filter level pruning method for deep neural network compression. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 5058–5066 (2017)
34. Malach, E., Yehudai, G., Shalev-Schwartz, S., Shamir, O.: Proving the lottery ticket hypothesis: pruning is all you need. In: III, H.D., Singh, A. (eds.) International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 119, pp. 6682–6691. PMLR (2020)
35. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11264–11272 (2019)
36. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. In: International Conference on Learning Representations (ICLR) (2017)

37. Morcos, A., Yu, H., Paganini, M., Tian, Y.: One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In: Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc. (2019)
38. Mosek, A.: The mosek optimization software. **54**(2–1), 5 (2010). www.mosek.com
39. Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., Srebro, N.: The role of over-parametrization in generalization of neural networks. In: 7th International Conference on Learning Representations, ICLR (2019)
40. Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., Rastegari, M.: What's hidden in a randomly weighted neural network? In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11893–11902 (2020)
41. Salama, A., Ostapenko, O., Klein, T., Nabi, M.: Pruning at a glance: global neural pruning for model compression. arXiv preprint arXiv:1912.00200 (2019)
42. Serra, T., Kumar, A., Ramalingam, S.: Lossless compression of deep neural networks. In: 2020 Fall Eastern Virtual Sectional Meeting, AMS (2020)
43. Shrikumar, A., Greenside, P., Kundaje, A.: Learning important features through propagating activation differences. In: International Conference on Machine Learning, pp. 3145–3153. PMLR (2017)
44. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Bengio, Y., LeCun, Y. (eds.) International Conference on Learning Representations (ICLR) (2015)
45. Srinivas, S., Babu, R.V.: Data-free parameter pruning for deep neural networks. arXiv preprint arXiv:1507.06149 (2015)
46. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural Netw. Mach. Learn. **4**(2), 26–31 (2012)
47. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: International Conference on Learning Representations (ICLR) (2019)
48. Tsay, C., Kronqvist, J., Thebelt, A., Misener, R.: Partition-based formulations for mixed-integer optimization of trained relu neural networks. Adv. Neural. Inf. Process. Syst. **34**, 3068–3080 (2021)
49. Verma, S., Pesquet, J.C.: Sparsifying networks via subdifferential inclusion. In: International Conference on Machine Learning, pp. 10542–10552. PMLR (2021)
50. Wang, C., Grosse, R., Fidler, S., Zhang, G.: EigenDamage: structured pruning in the kronecker-factored eigenbasis. In: International Conference on Machine Learning, pp. 6566–6575. PMLR (2019)
51. Wang, C., Zhang, G., Grosse, R.B.: Picking winning tickets before training by preserving gradient flow. In: International Conference on Learning Representations (ICLR) (2020)
52. Wang, Y., et al.: Pruning from scratch. In: AAAI, pp. 12273–12280 (2020)
53. Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., Liu, Q.: Good subnetworks provably exist: pruning via greedy forward selection. In: International Conference on Machine Learning, pp. 10820–10830. PMLR (2020)
54. Yu, R., et al.: NISP: pruning networks using neuron importance score propagation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9194–9203 (2018)
55. Yu, X., Serra, T., Ramalingam, S., Zhe, S.: The combinatorial brain surgeon: pruning weights that cancel one another in neural networks. In: International Conference on Machine Learning, pp. 25668–25683. PMLR (2022)

56. Zeng, W., Urtasun, R.: MLPrune: multi-layer pruning for automated neural network compression. In: International Conference on Learning Representations (ICLR) (2018)
57. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. In: International Conference on Learning Representations (ICLR) (2017)
58. Zhu, Z., et al.: A geometric analysis of neural collapse with unconstrained features. Adv. Neural. Inf. Process. Syst. **34**, 29820–29834 (2021)