The Combinatorial Brain Surgeon: Pruning Weights That Cancel One Another in Neural Networks

Xin Yu [*1] **Thiago Serra** [*2] **Shandian Zhe** [1] **Srikumar Ramalingam** [3]

Abstract

Neural networks tend to achieve better accuracy with training if they are larger—even if the resulting models are overparameterized. Nevertheless, carefully removing such excess of parameters before, during, or after training may also produce models with similar or even improved accuracy. In many cases, that can be curiously achieved by heuristics as simple as removing a percentage of the weights with the smallest absolute value—even though magnitude is not a perfect proxy for weight relevance. With the premise that obtaining significantly better performance from pruning depends on accounting for the combined effect of removing multiple weights, we revisit one of the classic approaches for impact-based pruning: the Optimal Brain Surgeon (OBS). We propose a tractable heuristic for solving the combinatorial extension of OBS, in which we select weights for simultaneous removal, as well as a systematic update of the remaining weights. Our selection method outperforms other methods under high sparsity, and the weight update is advantageous even when combined with the other methods.

1. Introduction

In a world where large and overparameterized neural networks keep GPUs burning hot because machine learning researchers rethought generalization (Zhang et al., 2017; Belkin et al., 2019) and started tuning neural architectures with a hope for globally convergent loss landscapes (Li et al., 2018; Sun et al., 2020), network pruning can perhaps save us from parameter redundancy (Denil et al., 2013).

Network pruning can lead to more parameter-efficient networks, with which we can save on model deploying and storage costs, and even produce models with better accuracy. Although the extent to which we can prune depends on the task (Liebenwein et al., 2021) and pruning may unevenly impact accuracy across classes if not carefully performed (Hooker et al., 2019; Paganini, 2020; Hooker et al., 2020), pruning can also make neural networks more robust against adversarial manipulation (Wu & Wang, 2021).

From a perspective of model expressiveness using linear regions, we may also see network pruning as a means to close the gap between the highly complex models that can be theoretically learned with an architecture (Pascanu et al., 2014; Montúfar et al., 2014; Telgarsky, 2015; Montúfar, 2017; Arora et al., 2018a; Serra et al., 2018; Serra & Ramalingam, 2020; Xiong et al., 2020; Montúfar et al., 2021) and the relatively less complex models obtained in practice (Hanin & Rolnick, 2019a;b; Tseran & Montúfar, 2021).

Curiously, however, the long-standing magnitude-based pruning approach (Hanson & Pratt, 1988; Mozer & Smolensky, 1989; Janowsky, 1989) remains remarkably competitive (Blalock et al., 2020) despite having equally long-standing evidence that it does not offer a good proxy for parameter relevance (LeCun et al., 1989). But why?

We conjecture that focusing on the impact of removing each parameter alone prevents impact-based methods from being more effective. In other words, we believe that the decision about pruning each parameter should be based on which other parameters are also removed. Ideally, we want the effect of such removals to cancel one another to the largest extent possible while achieving the aimed pruning rate.

In this work, we revisit the approach taken by classic methods such as Optimal Brain Damage (OBD) by LeCun et al. (1989) and Optimal Brain Surgeon (OBS) by Hassibi & Stork (1992) through the functional Taylor expansion of the loss function $\mathcal{L}$ on a choice of weights $\boldsymbol{w} \in \mathbb{R}^N$ around the learned weights $\bar{\boldsymbol{w}} \in \mathbb{R}^N$ of the neural network as

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}) - \mathcal{L}(\bar{\boldsymbol{w}}) =& \nabla\mathcal{L}(\bar{\boldsymbol{w}})(\boldsymbol{w} - \bar{\boldsymbol{w}}) \\ &+ \frac{1}{2}(\boldsymbol{w} - \bar{\boldsymbol{w}})^T \nabla^2 \mathcal{L}(\bar{\boldsymbol{w}})(\boldsymbol{w} - \bar{\boldsymbol{w}}) \\ &+ O(\|\boldsymbol{w} - \bar{\boldsymbol{w}}\|^3).\end{aligned}$$

Like OBD and OBS, we assume that (i) the training con-

[*]Equal contribution [1]School of Computing, University of Utah, Salt Lake City, UT, United States [2]Freeman College of Management, Bucknell University, Lewisburg, PA, United States [3]Google Research, New York, NY, United States. Correspondence to: Xin Yu <xin.yu@utah.edu>.

verged to a local minimum, so $\nabla \mathcal{L}(\bar{\boldsymbol{w}}) = 0$; and (ii) $\boldsymbol{w}$ is sufficiently close to $\bar{\boldsymbol{w}}$, so $O(\|\boldsymbol{w} - \bar{\boldsymbol{w}}\|^3) \approx 0$. Hence,

$$\mathcal{L}(\boldsymbol{w}) - \mathcal{L}(\bar{\boldsymbol{w}}) \approx \frac{1}{2}(\boldsymbol{w} - \bar{\boldsymbol{w}})^T \nabla^2 \mathcal{L}(\tilde{\boldsymbol{w}})(\boldsymbol{w} - \tilde{\boldsymbol{w}}). \quad (1)$$

Under such conditions, we can asses the impact of pruning the i-th weight with $w_i = 0$ and $w_j = \bar{w}_j \; \forall j \neq i$.

Local quadratic models of the loss function have been used in varied ways, then and now. In OBD, the Hessian matrix $\boldsymbol{H} := \nabla^2 \mathcal{L}(\tilde{\boldsymbol{w}})$ is further assumed to be diagonal, hence implying that pruning one weight has no impact on pruning the remaining weights. In OBS, the Hessian matrix $\boldsymbol{H}$ is no longer assumed to be diagonal, and the optimality conditions are used to update the remaining weights of the network based on removing the weight which causes the least approximate increase to the loss function. The first modern revival of this approach is the Layerwise OBS by Dong et al. (2017), in which each layer is pruned independently from the others. More recently, WoodFisher by Singh & Alistarh (2020) operates over all the layers by introducing a new method that more efficiently approximates the Hessian inverse $\boldsymbol{H}^{-1}$, which is necessary for the weight updates of OBS. In a sense, those modern revivals focused on keeping the calculation of $\boldsymbol{H}^{-1}$ manageable.

However, current OBS approaches do not account for the effect of one pruning decision on other pruning decisions. Whereas OBS does take into account the impact of pruning a given weight on the remaining weights, only the weight with smallest approximated impact on the loss function is pruned at each step. In the most recent work, Singh & Alistarh (2020) described the updates involved on choosing two weights for simultaneous removal, but nevertheless observed that considering the removal of multiple weights together would be impractical if performed in such a way.

While not disagreeing with Singh & Alistarh's stance, we nevertheless proceed to formulate this problem and then consider how to approach it in a tractable way. In a nutshell, the contributions of this paper are the following:

(i) We propose a formulation of the Combinatorial Brain Surgeon (CBS) problem using Mixed-Integer Quadratic Programming (MIQP), which for tractability is decomposed into the problems of pruned weight selection and unpruned weight update (Section 3);

(ii) We propose a local search algorithm to improve the solution from other methods for the selection of pruned weights, or CBS Selection (CBS-S) (Section 4);

(iii) We also propose a randomized extension of magnitude-based pruning to obtain a diverse pool of CBS-S solutions for the local search algorithm (Section 5); and

(iv) We extend the systematic weight update from OBS (Hassibi & Stork, 1992; Singh & Alistarh, 2020) for updating the unpruned weights subject to multiple pruned weights, or CBS Update (CBS-U) (Section 6).

We focus on single-shot pruning after training to make before and after comparisons easier; as well as to facilitate comparing the results of our method with existing work, in particular that of Singh & Alistarh (2020). We discuss additional related work in Section 2, evaluate the proposed algorithms in Section 7, and draw conclusions in Section 8).

2. Related Work

Blalock et al. (2020) observes that most work in network pruning relies on either magnitude-based or impact-based methods for selecting which weights to remove from the neural network. In fact, the list of key references for each is so long that we will use different paragraphs for each.

Magnitude-based methods select the weights with smallest absolute value (Hanson & Pratt, 1988; Mozer & Smolensky, 1989; Janowsky, 1989; Han et al., 2015; 2016; Li et al., 2017; Frankle & Carbin, 2019; Elesedy et al., 2020; Gordon et al., 2020; Tanaka et al., 2020; Liu et al., 2021b).

Impact-based methods aim to select weights which would have less impact on the model if removed. That includes gradient-based approaches such as ours but we also regard other approaches (LeCun et al., 1989; Hassibi & Stork, 1992; Hassibi et al., 1993; Lebedev & Lempitsky, 2016; Molchanov et al., 2017; Dong et al., 2017; Yu et al., 2018; Zeng & Urtasun, 2018; Baykal et al., 2019; Lee et al., 2019; Wang et al., 2019; Liebenwein et al., 2020; Wang et al., 2020; Xing et al., 2020; Singh & Alistarh, 2020).

To those we can add the recent stream of exact methods, which aim to preserve the model intact while pruning the network (Serra et al., 2020; Sourek & Zelezny, 2021; Serra et al., 2021; Chen et al., 2021; Anonymous, 2022). To the best of our understanding, these methods are currently only applicable to certain architectures or training algorithms.

In addition to the discussion of pruning two parameters by Singh & Alistarh (2020), other recent works implement joint parameter pruning approaches in neural networks. Chen et al. (2021) partition the parameters into zero-invariant groups for removal. Liu et al. (2021a) identify coupled channels that should be either kept or pruned together.

Across all these types of approaches, most work has been done on pruning trained neural networks and then fine tuning them afterwards. However, there is an growing body of work on pruning during training or at initialization (Frankle & Carbin, 2019; Lee et al., 2019; Liu et al., 2019; Lee et al., 2020; Wang et al., 2020; Renda et al., 2020; Tanaka et al., 2020; Frankle et al., 2021; Zhang et al., 2021).

The interest for the latter topic is in part attributed to the Lottery Ticket Hypothesis (LTH), according to which randomly initialized dense neural networks contain subnetworks that can be trained to achieve similar accuracy as the original network (Frankle & Carbin, 2019). That also sparked another stream of work on the strong LTH, according to which one can improve the accuracy of randomly initialized models by pruning instead of training (Zhou et al., 2019; Ramanujan et al., 2020; Malach et al., 2020; Pensia et al., 2020; Orseau et al., 2020; Qian & Klabjan, 2021; Chijiwa et al., 2021).

Another common theme is the formulation of optimization models for network pruning, which include references not mentioned above (He et al., 2017; Luo et al., 2017; Aghasi et al., 2017; ElAraby et al., 2020; Ye et al., 2020; Verma & Pesquet, 2021; Ebrahimi & Klabjan, 2021).

There are also more general types of approaches that would be better described as compression than as pruning methods, such as combining neurons and low-rank approximation, factorization, and random projection of weight matrices (Jaderberg et al., 2014; Denton et al., 2014; Lebedev et al., 2015; Srinivas & Babu, 2015; Mariet & Sra, 2016; Arora et al., 2018b; Wang et al., 2018; Su et al., 2018; Wang et al., 2019; Suzuki et al., 2020a;b; Suau et al., 2020; Li et al., 2020).

3. Pruning as an Optimization Problem: the Combinatorial Brain Surgeon

We formulate the Combinatorial Brain Surgeon (CBS) as an optimization problem. This formulation is based on the local quadratic model of the loss function previously described.

For a compression rate $r \in (0, 1]$ corresponding to the fraction of the weights $\boldsymbol{w} \in \mathbb{R}^N$ of a neural network to be pruned, we formulate the problem of selecting the $\lceil rN \rceil$ weights to remove and the remaining $N - \lceil rN \rceil$ weights to update with minimum approximate loss. In order to model the loss, we use $\bar{w}$ to denote the weights of the trained neural network before pruning, as well as $\boldsymbol{H}$ to denote the Hessian matrix $\nabla^2 \mathcal{L}(\bar{\boldsymbol{w}})$ and $\boldsymbol{H}_{i,j}$ to denote the element at the i-th row and j-th column. That yields the following Mixed-Integer Quadratic Programming (MIQP) formulation:

$$\begin{aligned}
\min \quad & \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (w_i - \bar{w}_i) \boldsymbol{H}_{i,j} (w_j - \bar{w}_j) && (2)\\
\text{subject to} \quad & \sum_{i=1}^{N} y_i = \lceil rN \rceil && (3)\\
& y_i \rightarrow w_i = 0 \qquad \forall i \in \{1, \dots, N\} && (4)\\
& y_i \in \{0, 1\} \qquad \forall i \in \{1, \dots, N\} && (5)\\
& w_i \in \mathbb{R} \qquad \forall i \in \{1, \dots, N\} && (6)
\end{aligned}$$

The decision variables of this formulation are, for each weight i, the updated value w_i of that weight and a binary variable y_i denoting whether the weight is pruned or not.

In order to obtain a tractable approach to CBS, we consider two special cases of this formulation in what follows.

CBS Selection We first consider the CBS Selection (CBS-S) formulation, in which we aim to minimize the approximate loss from selecting the weights to be pruned without updating the remaining weights. In terms of the general formulation above, that means that $w_i = \bar{w}_i$ if $y_i = 0$ and $w_i = 0$ if $y_i = 1$. With each weight restricted to a binary domain, i.e., $w_i \in \{0, \bar{w}_i\}$, the terms of the objective function are nonzero only in the case of pairs of weights which are both removed. Hence, we can abstract the decision variables associated with weights and avoid implementing the indicator constraint in Equation (4), which otherwise could have lead to numerical difficulties. That yields the following Integer Quadratic Programming (IQP) formulation:

$$\begin{aligned}
\min \quad & \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \bar{w}_i y_i \boldsymbol{H}_{i,j} \bar{w}_j y_j && (7)\\
\text{subject to} \quad & \sum_{i=1}^{N} y_i = \lceil rN \rceil && (8)\\
& y_i \in \{0, 1\} \qquad \forall i \in \{1, \dots, N\} && (9)
\end{aligned}$$

The formulation above is at the core of how we identify a combination of pruned weights affecting the local approximation of the loss function by the least amount. Namely, the impact of pruning both weights i and j is captured by $\frac{1}{2} \left(w_i \boldsymbol{H}_{i,j} w_j + w_j \boldsymbol{H}_{j,i} w_i \right)^\dagger$. In other words, the impact of pruning a weight depends on the other pruned weights. If weight updates are not considered, CBS-S is all you need.

Note that it is possible to linearize this formulation by replacing $y_i y_i$ with y_i and using a binary decision variable $z_{i,j}$ in place of each term $y_i y_j, i \neq j$, with the use of additional constraints $z_{i,j} \leq y_i$, $z_{i,j} \leq y_j$, and $z_{i,j} \geq y_i + y_j - 1$ (Padberg, 1989). However, that would lead to a quadratic increase in the number of decision variables and constraints, which would be impractical for large neural networks.

CBS Update We next consider the CBS Update (CBS-U) formulation, in which we aim to minimize the approximate loss from updating the remaining weights. We assume a selection of pruned weights as a solution $\tilde{\boldsymbol{y}}$ for CBS-S. Hence, we abstract the binary decision variables from CBS by constraining that $w_i = 0$ when $\tilde{y}_i = 0$. That yields the

†For generality, we do not assume $\boldsymbol{H}$ to be symmetric.

following Quadratic Programming (QP) formulation:

$$\min \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(w_i - \bar{w}_i)\boldsymbol{H}_{i,j}(w_j - \bar{w}_j) \tag{10}$$

$$\text{subject to} \quad w_i = 0 \qquad \forall i \in \{1, \ldots, N\} : \tilde{y}_i = 1 \tag{11}$$

$$w_i \in \mathbb{R} \qquad \forall i \in \{1, \ldots, N\} : \tilde{y}_i = 0 \tag{12}$$

By abstracting the pruned weights altogether, we can reformulate CBS-U as an unconstrained quadratic optimization problem which can be efficiently solved (see Section 6).

Dissociating CBS into two subproblems has consequences, good and bad. On the one hand, solving each subproblem to optimality does not necessarily imply that we would obtain an optimal solution to CBS. However, it is very unlikely that we would obtain an optimal solution to CBS-S for neural networks of reasonable size in the first place. If we were to nevertheless contemplate such a possibility, we could use a Benders-type decomposition (Benders, 1962; Hooker & Ottosson, 2003) through which we would alternate between solving CBS-S and CBS-U by iteratively adding constraints to an extension of the CBS-S formulation to account for the change to the loss function due to weight updates for the selections of pruned weights so far obtained with CBS-S.

On the other hand, this dissociation of the weight selection and weight update problems leaves most of the computational difficulty to the former. This is particularly beneficial because it allows us to explore tried-and-tested techniques commonly used to solve discrete optimization problems.

Although CBS-S—and even CBS—could potentially be fed into Mixed-Integer Programming (MIP) solvers capable of producing optimal solutions, that would not scale to neural networks of reasonable size. We refer the reader interested in formulations and algorithms for MIP problems to Conforti et al. (2014). Interestingly, the algorithmic improvements of MIP solvers have been comparable to their contemporary hardware improvements for decades (Bixby, 2012), but before that happened the tricks of the trade were different: they involved designing good heuristics. In the next sections, we will resort to heuristic techniques that allowed optimizers to obtain good solutions for seemingly intractable problems subject to the solvers and hardware of their time—and even to those of today in cases like ours.

4. A Greedy Swapping Local Search Algorithm to Improve Pruning Selection

We begin to describe our approach by the local search, which is where we take full advantage of the interdependence between pruned weights. Local search methods are used to produce better solutions or a diversified pool of solutions based on adjusting an initial solution, which in our case would be a selection of weights $\mathbb{P}$ to be pruned. We refer the reader interested in local search to Aarts & Lenstra (1996).

In particular, our local search method iteratively swaps a weight in $\mathbb{P}$ with a weight not in $\mathbb{P}$. Similar operations have been long used for the traveling salesperson problem (Applegate et al., 2006), in which the 2-opt (Croes, 1958) and the 3-opt (Lin, 1965) algorithms respectively swap the arcs among two and three vertices until the solution is no further improved. In our case, since minimizing the loss function on the training set may not necessarily lead to an improvement on test set accuracy, we found that avoiding swaps with negligible loss improvement lead to better test accuracy.

Algorithm 1 describes our local search method. The outer loop repeats for $steps_{max}$ steps, unless the sample loss is not improved in $noimp_{max}$ consecutive steps or a step concludes without changing the set of pruned weights $\mathbb{P}$. At every repetition of the outer loop, we initialize the sets $\mathbb{I} \subseteq \mathbb{P}$ and $\mathbb{J} \subseteq \bar{\mathbb{P}} := \{1, \ldots, N\} \setminus \mathbb{P}$ that will respectively keep track of the weights in $\mathbb{P}$ that are no longer pruned and the weights in $\bar{\mathbb{P}}$ that are pruned in lieu of those in $\mathbb{I}$. For each weight $i \in \mathbb{P}$, we calculate α_i in Line 7 as the impact of pruning weight i if the other pruned weights are also those in $\mathbb{P}$. The weights in $\mathbb{P}$ are then sorted in a sequence π of nonincreasing α values in Line 9, hence from largest to the smallest impact. For each weight $j \in \bar{\mathbb{P}}$, we calculate β_j in Line 11 as the impact of having weight j removed in addition to all the weights in $\mathbb{P}$. The weights in $\bar{\mathbb{P}}$ are then sorted in a sequence θ of nondecreasing impact if swapped with the first element $i := \pi_1$ in Line 13, hence leading to the first element $j := \theta_1$ yielding the greatest reduction if swapped with i. If swapping i and j does not yield a reduction of at least ε, the local search stops in Line 15. Otherwise, we loop with variable i over the currently pruned weighs in $\mathbb{P}$ and with variable j over the unpruned weights between Line 18 and Line 29. The weights in $\mathbb{P}$ are visited according to the sequence π, and for the element at the ii-th position of π we consider only the elements of θ between positions $ii - \rho$ and $ii + \rho$. The loop is interrupted if we are unsuccessful in swapping out τ parameters of $\mathbb{P}$. Parameter ε is used in Line 14 and Line 21 to restrict changes to the pruning set $\mathbb{P}$ to cases in which the local estimate of the loss function improves by at lest ε. To simplify notation, we do not divide all the calculated impacts by 2.

We refer to Appendix A for the matrix operations to calculate α, β, and γ with GPUs; and we refer to Appendix B for how to approximate and decompose the Hessian matrix $\boldsymbol{H}$ in order to reduce time and space complexity.

5. A Greedy Randomized Constructive Algorithm for Pruning Selection

As a typically local convergent method, the success of local search depends on the quality of the initial solution. We

Algorithm 1 Prune Selection Swapping Local Search

1: **Input:** $\mathbb{P}$ - initial set of pruned weights, ε - min impact variation for changing $\mathbb{P}$, τ - max failed weight swap attempts in $\mathbb{P}$, ρ - range of candidates for each weight swap, $steps_{max}$ - max number of steps, $noimp_{max}$ - max number of non-improving steps
2: **Output:** updated set of pruned weights $\mathbb{P}_F$
3: Initialize $\mathbb{P}_F \leftarrow \mathbb{P}, sImprove \leftarrow 0,$
4: **for** $s \leftarrow 1$ **to** $steps_{max}$ **do**
5: $\quad \mathbb{I} \leftarrow \emptyset; \mathbb{J} \leftarrow \emptyset$
6: $\quad$ **for** $i \in \mathbb{P}$ **do**
7: $\quad\quad \alpha_i \leftarrow \bar{w}_i \boldsymbol{H}_{i,i} \bar{w}_i + \sum_{j \in \mathbb{K}: j \neq i} (\bar{w}_i \boldsymbol{H}_{i,j} \bar{w}_j + \bar{w}_j \boldsymbol{H}_{j,i} \bar{w}_i)$
8: $\quad$ **end for**
9: $\quad$ Compute a sequence π of $i \in \mathbb{P}$ by nonincreasing α_i
10: $\quad$ **for** $j \in \bar{\mathbb{P}} := \{1, \ldots, N\} \setminus \mathbb{P}$ **do**
11: $\quad\quad \beta_j \leftarrow \boldsymbol{w}_j \boldsymbol{H}_{j,j} \boldsymbol{w}_j + \sum_{i \in \mathbb{K}} (\boldsymbol{w}_i \boldsymbol{H}_{i,j} \boldsymbol{w}_j + \boldsymbol{w}_j \boldsymbol{H}_{j,i} \boldsymbol{w}_i)$
12: $\quad$ **end for**
13: $\quad$ Compute a sequence θ of $j \in \bar{\mathbb{P}}$ by nondecreasing $\beta_j - \gamma_{\pi_1,j}$, where $\gamma_{i,j} := w_i \boldsymbol{H}_{i,j} w_j + w_j \boldsymbol{H}_{j,i} w_i$
14: $\quad$ **if** $\beta_{\theta_1} - \gamma_{\pi_1,\theta_1} - \alpha_{\pi_1} > -\varepsilon$ **then**
15: $\quad\quad$ **Terminate**
16: $\quad$ **end if**
17: $\quad c = 0$
18: $\quad$ **for** $i \in [\pi_1, \ldots, \pi_{|\mathbb{P}|}]$ **do**
19: $\quad\quad ii \leftarrow$ index of i in π; $c \leftarrow c + 1$
20: $\quad\quad$ **for** $j \in [\theta_{\max\{1, ii-\rho\}}, \ldots, \theta_{\min\{|\bar{\mathbb{P}}|, ii+\rho\}}] \setminus \mathbb{J}$ **do**
21: $\quad\quad\quad$ **if** $\left(\beta_j + \sum_{j' \in J} \gamma_{j,j'} - \sum_{i' \in I} \gamma_{i',j}\right) - (\gamma_{i,j}) - \left(\alpha_i + \sum_{j' \in J} \gamma_{i,j'} - \sum_{i' \in I} \gamma_{i,i'}\right) \leq -\varepsilon$ **then**
22: $\quad\quad\quad\quad \mathbb{I} \leftarrow \mathbb{I} + i; \mathbb{J} \leftarrow \mathbb{J} + j; c \leftarrow c - 1$
23: $\quad\quad\quad\quad$ **Goto Line 26**
24: $\quad\quad\quad$ **end if**
25: $\quad\quad$ **end for**
26: $\quad\quad$ **if** $c \geq \tau$ **then**
27: $\quad\quad\quad$ **Goto Line 30**
28: $\quad\quad$ **end if**
29: $\quad$ **end for**
30: $\quad$ **if** $\mathbb{I} = \emptyset$ **then**
31: $\quad\quad$ **Terminate**
32: $\quad$ **end if**
33: $\quad \mathbb{P} \leftarrow \mathbb{P} \cup J \setminus I$
34: $\quad$ **if** $\mathcal{L}(\mathbb{P}) < \mathcal{L}(\mathbb{P}_F)$ **then**
35: $\quad\quad \mathbb{P}_F \leftarrow \mathbb{P}; sImprov \leftarrow s$
36: $\quad$ **else if** $(s - sImprov) > noimp_{max}$ **then**
37: $\quad\quad$ **Terminate**
38: $\quad$ **end if**
39: **end for**

know that magnitude-based pruning can be rather effective, although not always perfect, as such an initial step. Hence, we consider how to leverage the guidance of magnitude-based pruning with the tricks of old-school optimizers.

In particular, we propose a constructive heuristic algorithm along the lines of semi-greedy methods (Hart & Shogan, 1987; Feo & Resende, 1989; 1995). These methods rely on a metric for identifying elements that are generally associated with good solutions. In our case, we know that selecting the weights with the smallest absolute value tends to produce good pruned models. Nevertheless, some randomness is introduced in the actual selection of elements, by which we may not necessarily always obtain a solution with the top-ranked elements for the chosen metric. By repeating this procedure a sufficient number of times, we are able to sample several solutions that closely follow the metric rather than a single solution that follows it rather strictly.

Algorithm 2 describes our constructive method. The outer loop from Line 4 to Line 7 produces S selections of weights to prune, among which the one with smallest sample loss is chosen. The inner loop from Line 12 to Line 16 iteratively partitions the weights of the neural network into B buckets of roughly the same size, and in which of those we select the same proportion of weights to be pruned. The partitioning step introduces randomness to the weight selection while still making the weights with smallest absolute value more likely to be pruned. This construction is highly parallelizable because the processing of each bucket can be done by a separate unit, hence favoring the use of modern GPUs.

Algorithm 2 Greedy Randomized Pruning Selection

1: **Input 1:** number of weight-partitioning buckets B
2: **Input 2:** number of pruning sets S to be generated
3: **Output:** best set of pruned weights $\mathbb{P}$ found
4: **for** $s \leftarrow 1$ **to** S **do**
5: $\quad \mathbb{Q} \leftarrow \emptyset$
6: $\quad \mathbb{W} \leftarrow \{1, \ldots, N\}$
7: $\quad$ **for** $k \leftarrow 1$ **to** B **do**
8: $\quad\quad \mathbb{B} \leftarrow$ Randomly pick $\min\{\lceil \frac{N}{B} \rceil, |\mathbb{W}|\}$ weights from $\mathbb{W}$
9: $\quad\quad \mathbb{W} \leftarrow \mathbb{W} \setminus \mathbb{B}$
10: $\quad\quad \mathbb{B}' \leftarrow$ Magnitude-based pruning selection on $\mathbb{B}$
11: $\quad\quad \mathbb{Q} \leftarrow \mathbb{Q} \cup \mathbb{B}'$
12: $\quad$ **end for**
13: $\quad$ **if** $s = 1$ **or** sample loss is smaller pruning $\mathbb{Q}$ than $\mathbb{P}$ **then**
14: $\quad\quad \mathbb{P} \leftarrow \mathbb{Q}$
15: $\quad$ **end if**
16: **end for**

6. The Systematic Weight Update for OBS with Multiple Pruned Weights

We resort to the same weight update technique described for one pruned weight by Hassibi & Stork (1992) and for two pruned weights by Singh & Alistarh (2020). To facilitate the parallel, we follow the notation of prior work as close as possible, starting with the perturbation $\delta \boldsymbol{w} = \boldsymbol{w} - \bar{\boldsymbol{w}}$. We dualize the constraint of CBS-U with the Lagrange multiplier λ to obtain the following Lagrangian $L(\boldsymbol{w}, \lambda)$:

$$L(\delta \boldsymbol{w}, \boldsymbol{\lambda}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \delta w_i \boldsymbol{H}_{i,j} \delta w_j + \sum_{i \in \mathbb{P}} \lambda_i (\boldsymbol{e}_i^T \delta \boldsymbol{w} + \bar{w}_i)$$

We use the solution $\delta \boldsymbol{w}^*$ of $\nabla L(\delta \boldsymbol{w}, \lambda) = 0$ to obtain the Lagrange dual function $g(\lambda)$ for the infimum of $L(\delta \boldsymbol{w}, \lambda)$:

$$\nabla L(\boldsymbol{w}, \boldsymbol{\lambda}) = \boldsymbol{H} \delta \boldsymbol{w}^* + \sum_{i \in \mathbb{P}} \lambda_i \boldsymbol{e}_i = 0$$
$$\rightarrow \delta \boldsymbol{w}^* = -\sum_{i \in \mathbb{P}} \lambda_i \boldsymbol{H}^{-1} \boldsymbol{e}_i$$

$$\begin{aligned} g(\boldsymbol{\lambda}) &= \frac{1}{2}\left(\sum_{i \in \mathbb{P}} \lambda_i \boldsymbol{H}^{-1} \boldsymbol{e}_i\right)^T \boldsymbol{H} \left(\sum_{i \in \mathbb{P}} \lambda_i \boldsymbol{H}^{-1} \boldsymbol{e}_i\right) \\ &\quad + \sum_{i \in \mathbb{P}} \lambda_i \left(-e_i^T \sum_{j \in \mathbb{P}} \lambda_j H^{-1} e_j + \bar{w}_i\right) \\ &= \frac{1}{2}\sum_{i \in \mathbb{P}}\sum_{j \in \mathbb{P}} \lambda_i \lambda_j e_i^T H^{-1} e_j \\ &\quad - \sum_{i \in \mathbb{P}}\sum_{j \in \mathbb{P}} \lambda_i \lambda_j e_i^T H^{-1} e_j + \sum_{i \in \mathbb{P}} \lambda_i \bar{w}_i \\ &= -\frac{1}{2}\sum_{i \in \mathbb{P}}\sum_{j \in \mathbb{P}} \lambda_i \lambda_j e_i^T H^{-1} e_j + \sum_{i \in \mathbb{P}} \lambda_i \bar{w}_i \end{aligned}$$

We then obtain the maximizer $\boldsymbol{\lambda}^*$ of the Lagrange dual function $g(\boldsymbol{\lambda})$ by solving the system $\nabla g(\boldsymbol{\lambda}) = 0$:

$$\nabla_{\lambda_i} g(\boldsymbol{\lambda}) = -\sum_{j \in \mathbb{P}} \lambda_j^* e_i^T \boldsymbol{H}^{-1} e_j + \bar{w}_i = 0 \qquad \forall i \in \mathbb{P}$$

Hence, if we denote by $[\boldsymbol{H}^{-1}]_{\mathbb{P},\mathbb{P}}$ the submatrix of $\boldsymbol{H}^{-1}$ on the rows and columns of the pruned weights $\mathbb{P}$, then the updated vectors of weights $\boldsymbol{w}^*$ minimizing the approximate loss function with the weights in $\mathbb{P}$ pruned is the following:

$$\boldsymbol{w}^* = \boldsymbol{\delta w}^* - \bar{\boldsymbol{w}} = -[\boldsymbol{H}^{-1}]_{\mathbb{P},\mathbb{P}} \boldsymbol{\lambda}^* - \bar{\boldsymbol{w}}$$

7. Experimental Evaluation

To validate our approach, we have applied it to compress commonly used networks for image classification. Given a well-trained model, we obtain sparser models and then we compare them with those obtained by both a state-of-the-art method as well as a commonly used method for unstructured pruning. Especially, we make detailed comparisons with WoodFisher (WF) (Singh & Alistarh, 2020), which is the state-of-the-art derived from OBS; including an adaption of WoodFisher, which we denote WoodFisher-S (WF-S), by which the remaining weights are not pruned and thus we restrict ourselves to pruning selection. We also include results for Magnitude-based Pruning (MP). We have included the Pytorch source code in Supplementary materials.

7.1. Experimental Setup

Although powerful for network pruning, the second-order approximation is expensive to apply on large networks. We make our algorithms more efficient by using the empirical Fisher and WoodFisher approximations of $\boldsymbol{H}$ and $\boldsymbol{H}^{-1}$.

Empirical Fisher As described in the methodology part, our algorithm requires the Hessian matrix $\boldsymbol{H}$ and its inverse $\boldsymbol{H}^{-1}$. In order to make them applicable to large networks, such as MobileNet (6.1M parameters), we use the empirical Fisher matrix to approximate $\boldsymbol{H}$. With such a decomposition of $\boldsymbol{H}$, if we randomly sample a subset of the training data of size K and the model has N parameters, the required space complexity is $O(K \times N)$ instead of $O(N^2)$. Singh & Alistarh (2020) note that a few hundred samples are sufficient for estimates applied to network pruning, and we have confirmed that while developing our approach. Furthermore, we can parallelize the tensor computation with such a decomposition and further speed up the local search algorithm. We refer to Appendix B for more details.

WoodFisher Approximation Due to the runtime cubic in the number of model parameters, it is inviable to directly compute $\boldsymbol{H}^{-1}$. Hence, we adopt the WoodFisher approximation by Singh & Alistarh (2020), which is an efficient method to estimate $\boldsymbol{H}^{-1}$ in $\mathcal{O}(K \times K)$ time and can be further reduced with a block-wise approximation in large networks. We use the same number of samples and block sizes as Singh & Alistarh (2020).

Pre-Trained Models We present our results on the MOBILENETV1 model of the STR method (Howard et al., 2017) trained on ImageNet (Deng et al., 2009), ResNet20 (He et al., 2016) and CifarNet model (Krizhevsky et al., 2009) (a revised AlexNet consisted of three convectional layers and two fully connected layers) trained on CIFAR10 (Krizhevsky et al., 2009), and MLPNet of three linear layers ($3072 \rightarrow 40 \rightarrow 20 \rightarrow 10$) trained on MNIST (LeCun et al., 1998). To make a fair comparison

Model	Fisher subsample size	Fisher mini-batch size	Chunk Size	Batch Size
MLPNet	1000	1	-	64
CIFARNet	1000	1	-	64
ResNet20	1000	1	-	64
MobileNet	400	2400	10000	128

Table 1. Parameters used for WoodFisher approximation of $\boldsymbol{H}^{-1}$. We unitize no chunking on MLPNet, block-wise estimation with respect to the layers without further chunking on CIFARNet and ResNet20.

with WoodFisher, we use the same checkpoints provided in its open-source implementation.

Local Search Parameters We use the following parameters for the local search algorithm: $\varepsilon = 1e-4$, $\tau = 20$, $\rho = 10$, $steps_{max} = 50$, and $noimp_{max} = 5$. We order the α and β at the beginning of local search. The selection of the parameters follows our observations that only minor changes will occur after a few pairs of weights are swapped.

Sparsity Rates We have initially experimented with sparsity rates ranging from 0.1 to 0.9 in multiples of 0.1. Having observed that in most cases the results across methods were very similar for lower rates, we have focused on higher rates provided that the models remained better than random guessing, i.e., more than 10% accuracy for at least one model for uniformly distributed datasets with 10 classes each.

7.2. CBS-S Benchmarking

Our first results compare our CBS-S approach, consisting of the greedy randomized constructive algorithm and the greedy swapping local search algorithm, with the results obtained with MP as well as with WoodFisher-S. These results consist of the leftmost part of Tables 2, 3, 4, and 5. In general, CBS-S produces more accurate models than MP and WF-S.

7.3. CBS Benchmarking

Our next results compare our CBS approach, consisting of CBS-S as well as the systematic weight update for CBS-U, with the results obtained with WoodFisher. These results consist of the rightmost part of Tables 2, 3, 4, and 5. In most of the cases, CBS produces more accurate models than WF.

7.4. CBS-U Benchmarking

Our final results compare the benefit of CBS-U alone by applying it to the prune selection of MP and WF-S in Table 6. Except for cases of very low sparsity, applying CBS-U to the pruned model improves its accuracy.

8. Conclusion

> *Optimization matters only when it matters. When it matters, it matters a lot, but until you know that it matters, don't waste a lot of time doing it.*
>
> Newcomer (1996)

We have introduced the Combinatorial Brain Surgeon (CBS), an approach to network pruning that accounts for the joint effect of pruning multiple weights of a neural network. Our approach is based on the same paradigm as the classic Optimal Brain Surgeon (OBS), in which the selection of pruned weights is also followed by the update of remaining weights and both aim to minimize a local quadratic approximation of the sample loss. We obtain a tractable approach to the optimization problem of CBS by dissociating the pruning selection and the weight update as distinct subproblems: CBS Selection (CBS-S) and CBS Update (CBS-U), respectively.

One benefit of this dissociation is that CBS Selection can be used as a stand-alone pruning technique if updating the remaining weights is not in the scope of the pruning algorithm, such as when pruning is performed at initialization or during training. Due to the presently large number of parameters in neural networks, CBS-S remains a challenging problem for a straightforward approach with modern optimization solvers. Nevertheless, we resort to tried-and-tested techniques that were once the best resort to tackle many discrete optimization problems: we design a local search method intended to leverage the synergy between pruned weights to minimize the sample loss approximation; and we design a greedy randomized constructive heuristic intended to leverage the effectiveness of magnitude-based pruning.

We have observed competitive results for our approach to CBS-S as well as to CBS by also extending the weight update from OBS to multiple parameters with CBS-U. More specifically, we obtain substantially more accurate models in both cases for higher sparsity rates, such as 0.90, 0.95, and 0.98. We believe that this is particularly encouraging, since the purpose of using network pruning is to reduce the number of parameters by the largest possible amount.

To the best of our understanding, this is the first paper that considers joint impact in network pruning at the parameter level. That comes with a rather challenging optimization problem, to which we provide a first approach. We hope to see it further improved in future work, as we believe that this is genuinely a case in which optimization matters a lot!

References

Aarts, E. and Lenstra, J. *Local Search in Combinatorial Optimization*. Wiley, 1996.

Aghasi, A., Abdi, A., Nguyen, N., and Romberg, J. Net-

	Prune Selection				Weight Update		
Sparsity	*MP*	*WF-S*	*CBS-S*	**Improvement**	*WF*	*CBS*	**Improvement**
0.50	93.93	93.92	93.91	-0.02	**94.02**	**93.96**	-0.06
0.70	93.62	93.48	93.75	0.13	**93.77**	**93.98**	0.21
0.90	90.30	90.77	**92.37**	1.60	91.69	**93.14**	1.45
0.95	83.64	83.16	**88.24**	4.60	85.54	**88.92**	3.38
0.98	32.25	34.55	**66.64**	32.09	38.26	**55.45**	17.20

Table 2. The pruning performacne of different methods on MLPNet trained on MNIST. The accuracy of the model before pruning is 93.97%. The results were averaged over five runs. The **best** and **second** best results are highlighted.

	Prune Selection				Weight Update		
Sparsity	*MP*	*WF-S*	*CBS-S*	**Improvement**	*WF*	*CBS*	**Improvement**
0.30	90.77	90.81	90.97	0.16	**91.37**	**91.35**	-0.01
0.50	88.44	88.06	89.32	0.88	**90.23**	**90.58**	0.35
0.60	85.24	84.95	86.48	1.24	**87.96**	**88.88**	0.92
0.70	78.79	78.09	79.55	0.76	**81.05**	**81.84**	0.79
0.80	54.01	52.05	**61.30**	7.29	**62.63**	51.28	-11.35
0.90	11.79	11.44	**16.83**	5.04	11.49	**13.68**	2.19

Table 3. The pruning performance of different methods on ResNet20 trained on Cifar10. The accuracy of the model before pruning is 91.36%. The results were averaged over five runs. The **best** and **second** best results are highlighted.

	Prune Selection				Weight Update		
Sparsity	*MP*	*WF-S*	*CBS-S*	**Improvement**	*WF*	*CBS*	**Improvement**
0.50	79.77	79.76	**79.84**	0.08	79.76	**79.82**	0.06
0.75	**79.20**	78.76	79.07	0.31	79.17	**79.35**	0.18
0.86	66.90	68.53	73.55	5.02	**74.04**	**75.64**	1.60
0.93	22.85	24.51	**42.05**	17.54	31.80	**34.41**	2.61
0.95	17.61	14.81	**32.63**	17.82	15.29	**21.07**	5.78
0.97	10.29	10.15	**19.18**	9.03	11.11	**13.77**	2.66
0.99	10.42	9.34	**15.45**	6.15	9.90	**12.16**	2.26

Table 4. The pruning performance of different methods on CifarNet trained on Cifar10. The accuracy of the model before pruning is 79.75%. The results were averaged over five runs. The **best** and **second** best results are highlighted.

	Prune Selection				Weight Update		
Sparsity	*MP*	*WF-S*	*CBS-S*	**Improvement**	*WF*	*CBS*	**Improvement**
0.30	71.60	71.68	71.48	-0.20	**71.88**	**71.88**	0.00
0.40	69.16	69.53	69.37	-0.16	**71.15**	**71.45**	0.30
0.50	62.61	63.10	62.96	-0.14	**68.91**	**70.21**	1.30
0.60	41.94	44.07	43.10	-0.97	**60.90**	**66.37**	5.47
0.70	6.78	7.33	7.63	0.3	**29.36**	**55.11**	25.75
0.80	0.11	0.11	0.12	0.01	**0.24**	**16.38**	16.14

Table 5. The pruning performance of different methods on MobileNet trained on ImageNet. The accuracy of the model before pruning is 72.0%. The **best** and **second** best results are highlighted.

	Magnitude-Based Pruning			WoodFisher		
Sparsity	*MP*	*MP + CBS-U*	**Improvement**	*WF*	*WF-S + CBS-U*	**Improvement**
0.20	72.04	71.99	-0.05	**72.01**	**72.00**	-0.01
0.30	71.61	**71.88**	0.27	**71.88**	71.86	-0.02
0.40	69.16	**71.43**	2.27	71.15	**71.43**	0.28
0.50	62.61	**70.24**	7.63	68.91	**70.30**	1.39
0.60	41.94	66.33	24.39	**60.90**	**66.62**	5.72
0.70	6.78	**55.51**	48.73	29.36	**56.09**	26.73
0.80	0.11	**16.58**	16.47	0.24	**17.94**	17.70

Table 6. Ablation study on CBS-U and comparision with the weight update of WoodFisher method. This is tested on MobileNet trained on ImageNet with the accuracy of 72.0% before pruning. The **best** and **second** best results are highlighted.

Trim: Convex pruning of deep neural networks with performance guarantee. In *NeurIPS*, 2017.

Anonymous. Model compression via symmetries of the parameter space. 2022. URL https://openreview.net/forum?id=8MN_GH4Ckp4.

Applegate, D., Bibxy, R., Chvátal, V., and Cook, W. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

Arora, R., Basu, A., Mianjy, P., and Mukherjee, A. Understanding deep neural networks with rectified linear units. In *ICLR*, 2018a.

Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. Stronger generalization bounds for deep nets via a compression approach. In *ICML*, 2018b.

Baykal, C., Liebenwein, L., Gilitschenski, I., Feldman, D., and Rus, D. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *ICLR*, 2019.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. Reconciling modern machine learning practice and the bias-variance trade-off. *PNAS*, 116(32):15849–15854, 2019.

Benders, J. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

Bixby, R. A brief history of linear and mixed-integer programming computation. In *Documenta Mathematica, Extra Volume: Optimization Stories*, pp. 107–121, 2012.

Blalock, D., Ortiz, J., Frankle, J., and Guttag, J. What is the state of neural network pruning? In *MLSys*, 2020.

Chen, T., Ji, B., Ding, T., Fang, B., Wang, G., Zhu, Z., Liang, L., Shi, Y., Yi, S., and Tu, X. Only train once: A one-shot neural network training and pruning framework. In *NeurIPS*, 2021.

Chijiwa, D., Yamaguchi, S., Ida, Y., Umakoshi, K., and Inoue, T. Pruning randomly initialized neural networks with iterative randomization. In *NeurIPS*, 2021.

Conforti, M., Cornuéjols, G., and Zambelli, G. *Integer Programming*. Springer, 2014.

Croes, G. A method for solving traveling-salesman problems. *Operations Research*, 6:791–812, 1958.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and Freitas, N. Predicting parameters in deep learning. In *NeurIPS*, 2013.

Denton, E., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *NeurIPS*, 2014.

Dong, X., Chen, S., and Pan, S. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *NeurIPS*, 2017.

Ebrahimi, A. and Klabjan, D. Neuron-based pruning of deep neural networks with better generalization using kronecker factored curvature approximation, 2021.

ElAraby, M., Wolf, G., and Carvalho, M. Identifying efficient sub-networks using mixed integer programming. In *OPT Workshop*, 2020.

Elesedy, B., Kanade, V., and Teh, Y. W. Lottery tickets in linear models: An analysis of iterative magnitude pruning, 2020.

Feo, T. A. and Resende, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

Feo, T. A. and Resende, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

Frankle, J., Dziugaite, G., Roy, D., and Carbin, M. Pruning neural networks at initialization: Why are we missing the mark? In *ICLR*, 2021.

Gordon, M., Duh, K., and Andrews, N. Compressing BERT: Studying the effects of weight pruning on transfer learning. In *Rep4NLP Workshop*, 2020.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

Han, S., Mao, H., and Dally, W. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *ICLR*, 2016.

Hanin, B. and Rolnick, D. Complexity of linear regions in deep networks. In *ICML*, 2019a.

Hanin, B. and Rolnick, D. Deep relu networks have surprisingly few activation patterns. In *NeurIPS*, 2019b.

Hanson, S. and Pratt, L. Comparing biases for minimal network construction with back-propagation. In *NeurIPS*, 1988.

Hart, J. P. and Shogan, A. W. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6(3):107–114, 1987.

Hassibi, B. and Stork, D. Second order derivatives for network pruning: Optimal Brain Surgeon. In *NeurIPS*, 1992.

Hassibi, B., Stork, D., and Wolff, G. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, 1993.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

Hooker, J. and Ottosson, G. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

Hooker, S., Courville, A., Clark, G., Dauphin, Y., and Frome, A. What do compressed deep neural networks forget? *arXiv*, 1911.05248, 2019.

Hooker, S., Moorosi, N., Clark, G., Bengio, S., and Denton, E. Characterising bias in compressed models. *arXiv*, 2010.03058, 2020.

Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. MobileNets: Efficient convolutional neural networks for mobile vision applications. 2017.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

Janowsky, S. Pruning versus clipping in neural networks. *Physical Review A*, 1989.

Krizhevsky, A. et al. Learning multiple layers of features from tiny images. 2009.

Lebedev, V. and Lempitsky, V. Fast ConvNets using group-wise brain damage. In *CVPR*, 2016.

Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., and Lempitsky, V. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In *ICLR*, 2015.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. In *NeurIPS*, 1989.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Lee, N., Ajanthan, T., and Torr, P. SNIP: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.

Lee, N., Ajanthan, T., Gould, S., and Torr, P. A signal propagation perspective for pruning neural networks at initialization. In *ICLR*, 2020.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. Pruning filters for efficient convnets. In *ICLR*, 2017.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets. In *NeurIPS*, 2018.

Li, J., Sun, Y., Su, J., Suzuki, T., and Huang, F. Understanding generalization in deep learning via tensor methods, 2020.

Liebenwein, L., Baykal, C., Lang, H., Feldman, D., and Rus, D. Provable filter pruning for efficient neural networks. In *ICLR*, 2020.

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

Lin, S. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44:2245–2269, 1965.

Liu, L., Zhang, S., Kuang, Z., Zhou, A., Xue, J.-H., Wang, X., Chen, Y., Yang, W., Liao, Q., and Zhang, W. Group fisher pruning for practical network compression. In *ICML*, 2021a.

Liu, S., Chen, T., Chen, X., Atashgahi, Z., Yin, L., Kou, H., Shen, L., Pechenizkiy, M., Wang, Z., and Mocanu, D. C. Sparse training via boosting pruning plasticity with neuroregeneration. In *NeurIPS*, 2021b.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. In *ICLR*, 2019.

Luo, J., Wu, J., and Lin, W. ThiNet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

Malach, E., Yehudai, G., Shalev-Shwartz, S., and Shamir, O. Proving the lottery ticket hypothesis: Pruning is all you need. In *ICML*, 2020.

Mariet, Z. and Sra, S. Diversity networks: Neural network compression using determinantal point processes. In *ICLR*, 2016.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. In *ICLR*, 2017.

Montúfar, G. Notes on the number of linear regions of deep neural networks. In *SampTA*, 2017.

Montúfar, G., Pascanu, R., Cho, K., and Bengio, Y. On the number of linear regions of deep neural networks. In *NeurIPS*, 2014.

Montúfar, G., Ren, Y., and Zhang, L. Sharp bounds for the number of regions of maxout networks and vertices of Minkowski sums, 2021.

Mozer, M. and Smolensky, P. Using relevance to reduce network size automatically. *Connection Science*, 1989.

Newcomer, J. M. Optimization: Your worst enemy. http://flounder.com/optimization.htm, 1996. Accessed: 2022-01-28.

Orseau, L., Hutter, M., and Rivasplata, O. Logarithmic pruning is all you need. In *NeurIPS*, 2020.

Padberg, M. The boolean quadric polytope: Some characteristics, facets and relatives. *Mathematical Programming*, 45:139–172, 1989.

Paganini, M. Prune responsibly. *arXiv*, 2009.09936, 2020.

Pascanu, R., Montúfar, G., and Bengio, Y. On the number of response regions of deep feedforward networks with piecewise linear activations. In *ICLR*, 2014.

Pensia, A., Rajput, S., Nagle, A., Vishwakarma, H., and Papailiopoulos, D. Optimal lottery tickets via Subset-Sum: Logarithmic over-parameterization is sufficient. In *NeurIPS*, 2020.

Qian, X. and Klabjan, D. A probabilistic approach to neural network pruning. In *ICML*, 2021.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network? In *CVPR*, 2020.

Renda, A., Frankle, J., and Carbin, M. Comparing rewinding and fine-tuning in neural network pruning. In *ICLR*, 2020.

Serra, T. and Ramalingam, S. Empirical bounds on linear regions of deep rectifier networks. In *AAAI*, 2020.

Serra, T., Tjandraatmadja, C., and Ramalingam, S. Bounding and counting linear regions of deep neural networks. In *ICML*, 2018.

Serra, T., Kumar, A., and Ramalingam, S. Lossless compression of deep neural networks. In *CPAIOR*, 2020.

Serra, T., Yu, X., Kumar, A., and Ramalingam, S. Scaling up exact neural network compression by ReLU stability. In *NeurIPS*, 2021.

Singh, S. P. and Alistarh, D. WoodFisher: Efficient second-order approximation for neural network compression. In *NeurIPS*, 2020.

Sourek, G. and Zelezny, F. Lossless compression of structured convolutional models via lifting. In *ICLR*, 2021.

Srinivas, S. and Babu, R. V. Data-free parameter pruning for deep neural networks. In *BMVC*, 2015.

Su, J., Li, J., Bhattacharjee, B., and Huang, F. Tensorial neural networks: Generalization of neural networks and application to model compression, 2018.

Suau, X., Zappella, L., and Apostoloff, N. Filter distillation for network compression. In *WACV*, 2020.

Sun, R., Li, D., Liang, S., Ding, T., and Srikant, R. The global landscape of neural networks: An overview. *IEEE Signal Processing Magazine*, 37(5):95–108, 2020.

Suzuki, T., Abe, H., Murata, T., Horiuchi, S., Ito, K., Wachi, T., Hirai, S., Yukishima, M., and Nishimura, T. Spectral-pruning: Compressing deep neural network via spectral analysis. In *IJCAI*, 2020a.

Suzuki, T., Abe, H., and Nishimura, T. Compression based bound for non-compressed network: Unified generalization error analysis of large compressible deep neural network. In *ICLR*, 2020b.

Tanaka, H., Kunin, D., Yamins, D., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NeurIPS*, 2020.

Telgarsky, M. Representation benefits of deep feedforward networks. 2015.

Tseran, H. and Montúfar, G. On the expected complexity of maxout networks. In *NeurIPS*, 2021.

Verma, S. and Pesquet, J.-C. Sparsifying networks via subdifferential inclusion. In *ICML*, 2021.

Wang, C., Grosse, R., Fidler, S., and Zhang, G. EigenDamage: Structured pruning in the Kronecker-factored eigenbasis. In *ICML*, 2019.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020.

Wang, W., Sun, Y., Eriksson, B., Wang, W., and Aggarwal, V. Wide compression: Tensor ring nets, 2018.

Wu, D. and Wang, Y. Adversarial neuron pruning purifies backdoored deep models. In *NeurIPS*, 2021.

Xing, X., Sha, L., Hong, P., Shang, Z., and Liu, J. Probabilistic connection importance inference and lossless compression of deep neural networks. In *ICLR*, 2020.

Xiong, H., Huang, L., Yu, M., Liu, L., Zhu, F., and Shao, L. On the number of linear regions of convolutional neural networks. In *ICML*, 2020.

Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., and Liu, Q. Good subnetworks provably exist: Pruning via greedy forward selection. In *ICML*, 2020.

Yu, R., Li, A., Chen, C., Lai, J., Morariu, V., Han, X., Gao, M., Lin, C., and Davis, L. NISP: Pruning networks using neuron importance score propagation. In *CVPR*, 2018.

Zeng, W. and Urtasun, R. MLPrune: Multi-layer pruning for automated neural network compression. 2018.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

Zhang, Z., Chen, X., Chen, T., and Wang, Z. Efficient lottery ticket finding: Less data is more. In *ICML*, 2021.

Zhou, H., Lan, J., Liu, R., and Yosinski, J. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *NeurIPS*, 2019.

A. Approximating the Hessian and Matrix Operations for Local Search

If we denote the matrix of all network weights by $\boldsymbol{W}$ and we index both $\boldsymbol{W}$ and $\boldsymbol{H}$ with subsets to denote submatrices, we can calculate parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma}$ through matrix operations that leverage the parallel processing of GPUs:

$$\begin{aligned}
\boldsymbol{\alpha} &= \boldsymbol{W}_{\mathbb{P}} \bigodot (\boldsymbol{H}_{\mathbb{P},\mathbb{P}} \boldsymbol{W}_{\mathbb{P}}) + \left(\boldsymbol{W}_{\mathbb{P}}^{T} \boldsymbol{H}_{\mathbb{P},\mathbb{P}}\right)^{T} \bigodot \boldsymbol{W}_{\mathbb{P}} - \boldsymbol{W}_{\mathbb{P}} \bigodot \operatorname{diag}(\mathrm{H}_{\mathbb{P},\mathbb{P}}) \bigodot \mathrm{W}_{\mathbb{P}} \\
\boldsymbol{\beta} &= \boldsymbol{W}_{\overline{\mathbb{P}}} \bigodot \left(\boldsymbol{H}_{\overline{\mathbb{P}},\mathbb{P}} \boldsymbol{W}_{\mathbb{P}}\right) + \left(\boldsymbol{W}_{\mathbb{P}}{}^{T} \boldsymbol{H}_{\mathbb{P},\overline{\mathbb{P}}}\right)^{T} \bigodot \boldsymbol{W}_{\overline{\mathbb{P}}} + \boldsymbol{W}_{\overline{\mathbb{P}}} \bigodot \operatorname{diag}(\mathrm{H}_{\overline{\mathbb{P}},\overline{\mathbb{P}}}) \bigodot \mathrm{W}_{\overline{\mathbb{P}}} \\
\boldsymbol{\gamma}_{\mathbb{P},\bar{\mathbb{P}}} &= \boldsymbol{W}_{\mathbb{P}} \bigodot (\boldsymbol{H}_{\mathbb{P},\bar{\mathbb{P}}} \boldsymbol{W}_{\bar{\mathbb{P}}}) + \left(\boldsymbol{W}_{\bar{\mathbb{P}}}^{T} \boldsymbol{H}_{\bar{\mathbb{P}},\mathbb{P}}\right)^{T} \bigodot \boldsymbol{W}_{\mathbb{P}}
\end{aligned}$$

B. Approximating and Decomposing H to Reduce Time and Space Complexity

As we know, $\boldsymbol{H} \in \mathcal{R}^{\mathcal{N}\times\mathcal{N}}$ will consume large memory, especially for large network, when calculating $\boldsymbol{\alpha}, \boldsymbol{\beta}$, and $\boldsymbol{\gamma}$. We will alleviate this by the following formulation.

In this method, we approach the Hessian matrix with the Fisher matrix

$$H \approx F = \frac{1}{\mathcal{N}} \sum_{n=1}^{\mathcal{N}} \nabla \mathcal{L}_n \nabla \mathcal{L}_n^T, \quad \nabla \mathcal{L}_n = \nabla \mathcal{L}\left(y_n, f\left(x_n; W\right)\right)$$

Let's describe $\nabla\mathcal{L}_n$ as $g^n \in \mathcal{R}^D$ and have $H \approx \frac{1}{\mathcal{N}} \sum_{n=1}^{\mathcal{N}} g^n \left(g^n\right)^T$. The common operation for α, β, γ are the multiplication on the sub-matrix of $\boldsymbol{H}$ and the sub-vector of $\boldsymbol{W}$. We reformulate them as follows,

$$\begin{aligned}
\boldsymbol{H}_{\mathbb{S}_1\mathbb{S}_2} \mathbf{w}_{\mathbb{S}_2} &= \frac{1}{\mathcal{N}} \sum_{n=1}^{\mathcal{N}} g_{\mathbb{S}_1}^n \left(g_{\mathbb{S}_2}^n\right)^T \boldsymbol{W}_{\mathbb{S}_2} = \frac{1}{\mathcal{N}} \sum_{n=1}^{\mathcal{N}} g_{\mathbb{S}_1}^n \left(\boldsymbol{W}_{\mathbb{S}2}^T g_{\mathbb{S}_2}^n\right)^T \\
\boldsymbol{W}_{\mathbb{S}_1}^T \boldsymbol{H}_{\mathbb{S}_1\mathbb{S}_2} &= \frac{1}{\mathcal{N}} \sum_{n=1}^{\mathcal{N}} \boldsymbol{W}_{\mathbb{S}_1}^T g_{\mathbb{S}_1}^n \left(g_{\mathbb{S}_2}^n\right)^T
\end{aligned}$$

The computation cost for $\boldsymbol{H}_{\mathbb{S}_1\mathbb{S}_2}\boldsymbol{W}_{\mathbb{S}_2}$ is $|\mathbb{S}_1| \times |\mathbb{S}_2| \times |\mathbb{S}_2|$ while it is $\mathcal{N} \times |\mathbb{S}_1| + |\mathbb{S}_2| \times |\mathbb{S}_2|$ after decomposing H. Furthermore, the first one needs $D \times D + D$ memory and the later one only hold the gradients and the weights in memory with the memory complexity of $\mathcal{N} \times D + D$.

With the above formulation, we can further reduce the FLOP and memory consumption for this algorithm. We can further parallize the later one to further speed it up.