Fast as CHITA: Neural Network Pruning with Combinatorial Optimization

Riade Benbaki [1] **Wenyu Chen** [1] **Xiang Meng** [1] **Hussein Hazimeh** [2] **Natalia Ponomareva** [2] **Zhe Zhao** [2]
Rahul Mazumder [1]

Abstract

The sheer size of modern neural networks makes model serving a serious computational challenge. A popular class of compression techniques overcomes this challenge by pruning or sparsifying the weights of pretrained networks. While useful, these techniques often face serious tradeoffs between computational requirements and compression quality. In this work, we propose a novel optimization-based pruning framework that considers the combined effect of pruning (and updating) multiple weights subject to a sparsity constraint. Our approach, CHITA, extends the classical Optimal Brain Surgeon framework and results in significant improvements in speed, memory, and performance over existing optimization-based approaches for network pruning. CHITA's main workhorse performs combinatorial optimization updates on a memory-friendly representation of local quadratic approximation(s) of the loss function. On a standard benchmark of pretrained models and datasets, CHITA leads to superior sparsity-accuracy tradeoffs than competing methods. For example, for MLPNet with only 2% of the weights retained, our approach improves the accuracy by 63% relative to the state of the art. Furthermore, when used in conjunction with fine-tuning SGD steps, our method achieves significant accuracy gains over state-of-the-art approaches. Our code is publicly available at: https://github.com/mazumder-lab/CHITA.

[1]MIT [2]Google Research. Correspondence to: Riade Benbaki <rbenbaki@mit.edu>, Wenyu Chen <wenyu@mit.edu>, Xiang Meng <mengx@mit.edu>, Hussein Hazimeh <hazimeh@google.com>, Natalia Ponomareva <nponomareva@google.com>, Zhe Zhao <zhezhao@google.com>, Rahul Mazumder <rahulmaz@mit.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

1. Introduction

Modern neural networks tend to use a large number of parameters (Devlin et al., 2018; He et al., 2016), which leads to high computational costs during inference. A widely used approach to mitigate inference costs is to prune or sparsify pre-trained networks by removing parameters (Blalock et al., 2020). The goal is to obtain a network with significantly fewer parameters and minimal loss in performance. This makes model storage and deployment cheaper and easier, especially in resource-constrained environments.

Generally speaking, there are two main approaches for neural net pruning: (i) magnitude-based and (ii) impact-based. Magnitude-based heuristic methods (e.g., Hanson & Pratt, 1988; Mozer & Smolensky, 1989; Gordon et al., 2020) use the absolute value of weight to determine its importance and whether or not it should be pruned. Since magnitude alone may not be a perfect proxy for weight relevance, alternatives have been proposed. To this end, impact-based pruning methods (e.g. LeCun et al., 1989; Hassibi & Stork, 1992; Singh & Alistarh, 2020) remove weights based on how much their removal would impact the loss function, often using second-order information on the loss function. Both of these approaches, however, may fall short of considering the *joint effect* of removing (and updating) multiple weights simultaneously. The recent method CBS (Combinatorial Brain Surgeon) (Yu et al., 2022) is an optimization-based approach that considers the joint effect of multiple weights. The authors show that CBS leads to a boost in the performance of the pruned models. However, CBS can be computationally expensive: it makes use of a local model based on the second-order (Hessian) information of the loss function, which can be prohibitively expensive in terms of runtime and/or memory (e.g., CBS takes hours to prune a network with 4.2 million parameters).

In this work, we propose CHITA (Combinatorial Hessian-free Iterative Thresholding Algorithm), an efficient optimization-based framework for network pruning at scale. Our approach follows earlier works that consider a local quadratic approximation of the loss function based on the second-order Hessian information. Different from earlier work however, we make use of a simple yet important observation with which we can avoid computing and storing the

Hessian matrix to solve the optimization problem (hence the name "Hessian-free" in CHITA)—this allows us to address large networks rather efficiently. Specifically, we propose an equivalent reformulation of the problem as an ℓ_0-constrained sparse linear regression problem with a data matrix $A \in \mathbb{R}^{n\times p}$, where p is the number of trainable parameters in the original model and $n \lesssim 10^3$ (usually, $p \gg n$) is the number of the sub-samples used in approximating the Hessian. Compared to state-of-the-art approaches that consider a dense $p \times p$ matrix approximation of the Hessian, our approach leads to a significant reduction in memory usage (up to 10^3 times for $p = 10^6$) without any approximation.

Furthermore, we propose a novel approach to minimize our ℓ_0 regression reformulation, leveraging active set strategies, better stepsize selection, and various methods to accelerate convergence on the selected support. Our proposed approach builds upon vanilla Iterative Hard Thresholding methods (IHT, Blumensath & Davies, 2009) commonly used in the sparse learning literature, and significantly improves it. For instance, our framework can prune a network with 4.2M parameters to 80% sparsity (i.e., 0.84M nonzero parameters) in less than a minute and using less than 20GB of RAM [1].

Since the local quadratic model approximates the loss function only in a small neighborhood of the current solution (Singh & Alistarh, 2020), we also explore a multi-stage algorithm that updates the local quadratic model during pruning (but without retraining) and solves a more constrained problem in each stage, going from dense weights to sparse ones. Our experiments show that the resulting pruned models have a notably better accuracy compared to that of our single-stage algorithm and other pruning approaches. Furthermore, when used in the gradual pruning setting (Gale et al., 2019; Singh & Alistarh, 2020; Blalock et al., 2020) where re-training between pruning steps is performed, our pruning framework results in notable performance gains compared to state-of-the-art unstructured pruning methods.

Contributions Our contributions can be summarized as follows:

- We propose CHITA, a discrete optimization based framework for network pruning based on local quadratic approximation(s) of the loss function. We propose an ℓ_0-constrained sparse regression reformulation that avoids the pitfalls of storing a large dense Hessian, resulting in a significant reduction in memory usage (we work with an $n \times p$ matrix instead of a $p \times p$ one, with often $n \ll p$).
- A key workhorse of CHITA is a novel IHT-based algorithm to obtain good solutions to the sparse regression formulation. Exploiting problem structure, we propose methods to accelerate convergence and improve pruning performance, such as a new and efficient stepsize selection scheme and rapidly updating weights on the support. This leads to up to 1000x runtime improvement compared to existing network pruning algorithms.
- We show performance improvements across various models and datasets. In particular, CHITA results in a 98% sparse (i.e., 98% of weights in dense model are set to zero) MLPNet with 90% test accuracy (3% reduction in test accuracy compared to the dense model), which is a significant improvement over the previously reported best accuracy (55%) by CBS. As an application of our framework, we use it for gradual pruning and observe notable performance gains against state-of-the-art gradual pruning approaches.

2. Problem Setup and Related Work

In this section we present the general setup with connections to related work—this lays the foundation for our proposed methods discussed in Section 3.

2.1. Problem Setup

Consider a neural network with an empirical loss function $\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N} \ell_i(w)$, where $w \in \mathbb{R}^p$ is the vector of trainable parameters, N is the number of data points (samples), and $\ell_i(w)$ is a twice-differentiable function on the i-th sample. Given a pre-trained weight vector $\bar{w} \in \mathbb{R}^p$, our goal is to set some parameters to zero and possibly update other weights while maintaining the original model's performance (e.g., accuracy) as much as possible. In mathematical terms, given a pre-trained weight $\bar{w}$ and a target level of sparsity $\tau \in (0, 1)$, we aim to construct a new weight vector $w \in \mathbb{R}^p$ that satisfies:

- The loss function evaluated at w is as close as possible to the loss before pruning: that is, $\mathcal{L}(w) \approx \mathcal{L}(\bar{w})$.
- The number of nonzero weights at w respects the sparsity budget[2]: that is, $\|w\|_0 \leq (1-\tau)p$.

Following earlier work LeCun et al. (1989); Hassibi & Stork (1992); Singh & Alistarh (2020), we start with a local quadratic approximation of $\mathcal{L}$ around the pre-trained weight $\bar{w}$:

$$\mathcal{L}(w) = \mathcal{L}(\bar{w}) + \nabla\mathcal{L}(\bar{w})^\top(w-\bar{w}) + \frac{1}{2}(w-\bar{w})^\top\nabla^2\mathcal{L}(\bar{w})(w-\bar{w}) + O(\|w-\bar{w}\|^3). \quad (1)$$

With certain choices of gradient and Hessian approximations $g \approx \nabla\mathcal{L}(\bar{w}), H \approx \nabla^2\mathcal{L}(\bar{w})$, and ignoring higher-order

[1] For reference, CBS and Woodfisher would run out of memory in similar circumstances.

[2] Here ℓ_0 norm $\|w\|_0$ denotes the number of nonzero in the vector w.

terms, the loss $\mathcal{L}$ can be locally approximated by:

$$Q_0(w) := \mathcal{L}(\bar{w}) + g^\top (w - \bar{w}) + \frac{1}{2}(w - \bar{w})^\top H (w - \bar{w}). \quad (2)$$

Pruning the local approximation $Q_0(w)$ of the network can be naturally formulated as an optimization problem to minimize $Q_0(w)$ subject to a cardinality constraint, i.e.,

$$\min_w \; Q_0(w) \qquad \text{s.t.} \quad \|w\|_0 \leq k. \quad (3)$$

For large networks, solving Problem (3) directly (e.g., using iterative optimization methods) is computationally challenging due to the sheer size of the $p \times p$ matrix H. In Section 3.1, we present a Hessian-free reformulation of Problem (3), which plays an important role in our scalable optimization approach.

2.2. Related Work

Impact-based pruning dates back to the work of LeCun et al. (1989) where the OBD (Optimal Brain Damage) framework is proposed. This approach, along with subsequent ones (Hassibi & Stork, 1992; Singh & Alistarh, 2020; Yu et al., 2022) make use of the local approximation given in (2). It is usually assumed (but not in our work) that $\bar{w}$ is a local optimum of the loss function, and therefore $g = 0$ and $\mathcal{L}(w) \approx \mathcal{L}(\bar{w}) + \frac{1}{2}(w - \bar{w})^\top H (w - \bar{w})$. Using this approximation and also a diagonal Hessian H, OBD (Optimal Brain Damage, LeCun et al. (1989)) searches for a single weight i to prune with minimal change in the loss function. If the i-th weight is pruned ($w_i = 0, w_j = \bar{w}_j \forall j \neq i$), then the loss function increases by $\delta\mathcal{L}_i = \frac{\bar{w}_i^2}{2\nabla^2\mathcal{L}(\bar{w})_{ii}}$. This represents a score for each weight, and is used to prune weights in decreasing order of their score. OBS (Optimal Brain Surgeon, Hassibi & Stork (1992)) extends this algorithm by relaxing the assumption of the Hessian being diagonal, and using the optimality conditions to update the un-pruned weights. The authors also propose using the empirical Fisher information matrix, as an efficient approximation to the Hessian matrix. Layerwise OBS (Dong et al., 2017) proposes to overcome the computational challenge of computing the full (inverse) Hessian needed in OBS by pruning each layer independently. Singh & Alistarh (2020) use block-diagonal approximations of the Hessian matrix, which they approximate by the empirical Fisher information matrix on a small subset of the training data ($n \ll N$):

$$\nabla^2\mathcal{L}(\bar{w}) \approx H = \frac{1}{n}\sum_{i=1}^{n} \nabla\ell_i(\bar{w}) \nabla\ell_i(\bar{w})^\top. \quad (4)$$

While these approaches explore different ways to make the Hessian computationally tractable, they all rely on the OBD/OBS framework of pruning a single weight, and do not to consider possible interactions that can arise when pruning multiple weights. To this end, Yu et al. (2022) propose CBS (Combinatorial Brain Surgeon) which is an algorithm to approximately solve (3). While CBS shows impressive improvements in the accuracy of the pruned model over prior work, it appears to operate with the full dense $p \times p$ Hessian H. This limits scalability both in compute time and memory utilization, as p is often in the order of millions and more.

2.2.1. Choices of gradient approximation g

As mentioned earlier, most previous work assumes that the pre-trained weights $\bar{w}$ is a local optimum of the loss function $\mathcal{L}$, and thus take the gradient $g = 0$. However, the gradient of the loss function of a pre-trained neural network may not be zero in practice due to early stopping (or approximate optimization) (Yao et al., 2007). Thus, the WoodTaylor approach (Singh & Alistarh, 2020) proposes to approximate the gradient by the stochastic gradient, using the same samples for estimating the Hessian. Namely,

$$g = \frac{1}{n}\sum_{i=1}^{n} \nabla\ell_i(\bar{w}). \quad (5)$$

2.2.2. One-shot and gradual pruning

Generally speaking, *one-shot pruning* methods (LeCun et al., 1989; Singh & Alistarh, 2020; Yu et al., 2022) can be followed by a few fine-tuning and re-training steps to recover some of the accuracy lost when pruning. Furthermore, recent work has shown that pruning and re-training in a gradual fashion (hence the name, *gradual pruning*) can lead to big accuracy gains (Han et al., 2015; Gale et al., 2019; Zhu & Gupta, 2018). The work of Singh & Alistarh (2020) further shows that gradual pruning, when used with well-performing one-shot pruning algorithms, can outperform state-of-the-art unstructured pruning methods. In this paper, we focus on the one-shot pruning problem and show how our pruning framework outperforms other one-shot pruning methods (see Section 4.1). We then show that, if applied in the gradual pruning setting, our pruning algorithm outperforms existing approaches (see Section 4.2), establishing new state-of-the-art on MobileNetV1 and ResNet50.

3. Our Proposed Framework: CHITA

In this section, we present our algorithmic framework CHITA (Combinatorial Hessian-free Iterative Thresholding Algorithm) for pruning a network to a specific sparsity level. We formulate sparse network pruning by considering ℓ_0-regression problem(s) and propose scalable algorithms. For example, we can address networks with size $p \approx 10^6, n \approx 10^3, k \approx 10^5$ in less than one minute and using less than 20GB of memory. Our basic single-stage algorithm is an improved and scalable variant of IHT to solve (3).

In Section 3.3, we propose a sequential version of the single-step method that we call the *multi-stage* framework—this sequentially refines the local quadratic approximation and optimizes it (under sparsity constraints). The multi-stage version results in further performance boosts over the single-stage version as shown in Section 4.

3.1. An ℓ_0-regression formulation

Our formulation is based on a simple but important observation that the Hessian approximation in (4) has a low-rank structure:

$$H = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(\bar{w}) \nabla \ell_i(\bar{w})^\top = \frac{1}{n} A^\top A \in \mathbb{R}^{p\times p}, \quad (6)$$

where $A = [\nabla \ell_1(\bar{w}), \ldots, \nabla \ell_n(\bar{w})]^\top \in \mathbb{R}^{n\times p}$ has rank at most n with $10^3 \geq n \ll p$, and is hence low-rank.

Using the outer-product expression in display (6) and the gradient expression (5), we note that problem (3) can be equivalently written as the following cardinality constrained least squares problem (in Hessian-free form):

$$\min_{w} \ \frac{1}{2}\|b - Aw\|^2 \qquad \text{s.t.} \qquad \|w\|_0 \leq k, \quad (7)$$

where $b := A\bar{w} - e \in \mathbb{R}^n$ and e is a vector of ones. Furthermore, to improve solution quality (see discussion below), we include a ridge-like regularizer of the form $\|w - \bar{w}\|^2$ to the objective in (7). This leads to the following problem:

$$\begin{aligned} \min_{w} \ & Q(w) := \frac{1}{2}\|b - Aw\|^2 + \frac{n\lambda}{2}\|w - \bar{w}\|^2 \\ \text{s.t.} \ & \|w\|_0 \leq k, \end{aligned} \quad (8)$$

where $\lambda \geq 0$ is a parameter controlling the strength of the ridge-like regularizer that controls the proximity of w to the pre-trained weight vector $\bar{w}$ (See below for additional discussion). Note that our algorithm actually applies to the general form of optimization Problem (8) with a general b and A. We can recover special instances discussed earlier with particular choices of the problem parameters. We note that the regression formulation (8) does not require us to compute or store the full Hessian matrix $H \in \mathbb{R}^{p\times p}$. As discussed in Section 3.2, we only need to operate on the low-rank matrix A throughout our algorithm—this results in substantial gains in terms of both memory consumption and runtime.

3.1.1. Ridge term and choices of λ

We observe empirically that the success of our final pruned model depends heavily on the accuracy of the quadratic approximation of the loss function. Since this approximation is local, it is essential to ensure that the weights w during the pruning process are sufficiently close to the initial weights $\bar{w}$. One way[3] to achieve this is by including a squared ℓ_2 penalty, also known as the ridge, on the difference $w - \bar{w}$. This regularizer controlling the proximity of w to $\bar{w}$ does not appear to have been explicitly considered in previous works (Hassibi & Stork, 1992; Singh & Alistarh, 2020; Yu et al., 2022) on pruning using local quadratic approximations. While not used explicitly, this ridge regularizer appears to be used *implicitly*: this is achieved by adding a small diagonal $\lambda_0 I$ to the Fisher matrix with the goal of obtaining an invertible Fisher matrix. The usefulness of the regularization term λ is further explored in Appendix B.2.1. We observe that a well-chosen ridge term can help improve the test accuracy on MLPNet by 3%.

3.1.2. Relation to prior work on ℓ_0-regression problems

There is a substantial literature on algorithms for solving ℓ_0-regularized linear regression problems—see for example (Tillmann et al., 2021) for a recent overview on this topic. We provide a brief overview of related work, but it is worth noting that its applicability in the context of network pruning appears to be new. Sparse network pruning presents several interesting challenges, and in particular, the problem-scale we consider here makes our work different from earlier works in ℓ_0-sparse linear regression. As mentioned earlier, our algorithm is motivated by the well-known iterative hard thresholding method Blumensath & Davies (2009). The IHT algorithm involves projecting the weights onto the feasible set after each gradient step. Bertsimas & Van Parys (2020); Hazimeh et al. (2022) proposed algorithms to solve sparse regression problems to optimality via branch-and-bound. Beck & Eldar (2013) explore coordinate descent-type algorithms that update one/two coordinates at a time. Hazimeh & Mazumder (2020) propose efficient algorithms based on coordinate descent and local combinatorial optimization that applies to the unconstrained $\ell_0\ell_2$-penalized regression problem. We refer the reader to Hazimeh et al. (2022); Bertsimas et al. (2016) and the review paper (Tillmann et al., 2021) for a more comprehensive discussion of related work.

In summary, earlier methods for ℓ_0-regularized linear regression are quite effective at discovering high-quality solutions to Problem (7) for small to medium-sized problems and require k to be sufficiently small for efficiency. However, these methods are not well-suited for tackling large network pruning problems (e.g, $p \sim 10^6$ and $k \sim 10^5$) due to slow convergence or expensive per-iteration cost. To address this issue, we propose novel approaches for scalability in Section 3.2. Additionally, we emphasize that Problem (8) arises from a local approximation of the true loss $\mathcal{L}$ around $\bar{w}$. Therefore, achieving a high-quality solution for Prob-

[3] Another way is to introduce a multi-stage procedure, as explained in Section 3.3.

lem (8) alone does not guarantee a pruned network with high accuracy. To this end, we study a multi-stage extension (see Section 3.3) that requires solving several problems of the form (8).

3.2. Our proposed algorithm for Problem (8)

We present the core ideas of our proposed algorithm for Problem (8), and discuss additional details in Appendix A.

Our optimization framework relies on the IHT algorithm (Blumensath & Davies, 2009; Bertsimas et al., 2016) that optimizes (8) by simultaneously updating the support and the weights. By leveraging the low-rank structure, we can avoid the computational burden of computing the full Hessian matrix, thus reducing computational cost.

The basic version of the IHT algorithm can be slow for problems with a million weights, such as the ones we consider in this work. To improve the computational performance of our algorithm we propose a new line search scheme. Additionally, we use use an active set strategy and schemes to update the weights on the nonzero weights upon support stabilization. Taken together, we obtain notable improvements in computational efficiency and solution quality over traditional IHT, making it a viable option for network pruning problems at scale.

3.2.1. A Structure-aware IHT update

The IHT algorithm operates by taking a gradient step of size τ from the current iteration, then projects it onto the set of points with a fixed number ofnonzero coordinates through hard thresholding. Specifically, for any vector x, let $\mathcal{I}_k(x)$ denote the indices of k components of x that have the largest absolute value. The hard thresholding operator $P_k(x)$ is defined as $y_i = x_i$ if $i \in \mathcal{I}_k(x)$, and $y_i = 0$ if $i \notin \mathcal{I}_k(x)$; where y_i is the i-th coordinate of $P_k(x)$. IHT applied to problem (8) leads to the following update:

$$\begin{aligned} w^{t+1} &= \mathsf{HT}(w^t, k, \tau^s) := P_k\left(w^t - \tau^s \nabla Q(w^t)\right) \qquad (9)\\ &= P_k\left(w^t - \tau^s (A^\top(Ab - w^t) + n\lambda(w^t - \bar{w}))\right), \end{aligned}$$

where $\tau^s > 0$ is a suitable stepsize. The computation of $\mathsf{HT}(w^t, k, \tau^s)$ involves only matrix-vector multiplications with A (or $A^\top$) and a vector, which has a total computation cost of $O(np)$. This is a significant reduction compared to the $O(p^2)$ cost while using the full Hessian matrix as Singh & Alistarh (2020); Yu et al. (2022) do.

Active set strategy. In an effort to further facilitate the efficiency of the IHT method, we propose using an active set strategy, which has been proven successful in various optimization contexts—see for example, the works Nocedal & Wright (1999); Friedman et al. (2010); Hazimeh & Mazumder (2020). Loosely speaking, this strategy works by restricting the IHT updates to an *active set* (i.e., a relatively small subset of variables) and occasionally augmenting the active set with variables that violate certain optimality conditions. By implementing this strategy, the iteration complexity of the algorithm can be reduced to $O(nk)$ in practice, resulting in an improvement, when k is smaller than p. The algorithm details can be found in Appendix A.3.

3.2.2. Determining a good stepsize

Choosing an appropriate stepsize τ^s is crucial for fast convergence of the IHT algorithm. To ensure convergence to a stationary solution, a common choice is to use a constant stepsize of $\tau^s = 1/L$ (Bertsimas et al., 2016; Hazimeh & Mazumder, 2020), where L is the Lipschitz constant of the gradient of the objective function. This approach, while reliable, can lead to conservative updates and slow convergence—refer to Appendix A.1 for details. An alternative method for determining the stepsize is to use a backtracking line search, as proposed in Beck & Teboulle (2009). The method involves starting with a relatively large estimate of the stepsize and iteratively shrinking the step size until a sufficient decrease of the objective function is observed. However, this approach requires multiple evaluations of the objective function, which can be computationally expensive.

Our novel scheme. We propose a novel line search method for determining the stepsize to improve the convergence speed of IHT. Specifically, we develop a method that (approximately) finds the stepsize that leads to the maximum decrease in the objective, i.e., we attempt to solve

$$\min_{\tau^s \geq 0} \; g(\tau^s) := Q\left(P_k\left(w^t - \tau^s \nabla Q(w^t)\right)\right). \qquad (10)$$

For general objective functions, solving the line search problem (as in (10)) is challenging. However, in our problem, we observe and exploit an important structure: $g(\tau^s)$ is a piecewise quadratic function with respect to τ^s. Thus, the optimal stepsize on each piece can be computed exactly, avoiding redundant computations (associated with finding a good stepsize) and resulting in more aggressive updates. In Appendix A.1, we present an algorithm that finds a good stepsize by exploiting this structure. Compared to standard line search, our algorithm is more efficient, as it requires fewer evaluations of the objective function and yields a stepsize that results in a steeper descent.

3.2.3. Additional techniques for scalability

While the IHT algorithm can be quite effective in identifying the appropriate support, its progress slows down considerably once the support is identified (Blumensath, 2012), resulting in slow convergence. To accelerate the algorithm, we propose two techniques that refine thenonzero coefficients on the identified support by solving sub-problems: (i) Coordinate Descent (CD, Bertsekas, 1997; Nesterov, 2012) that updates each nonzero coordinate (with others fixed)

as per a cyclic rule; (ii) Back-solve based on Woodbury formula (Max, 1950) that calculates the optimal solution exactly on a restricted set of size k. We found both (i), (ii) to be useful for improving the accuracy of the pruned network. Further details on the methods (i), (ii) are in Appendix A.2 and A.4.

Our single-stage algorithm CHITA glues together the different pieces discussed above into a coherent algorithm. It takes as input a low-rank matrix A, the initial weight $\bar{w}$ and the ℓ_0-constraint k; and returns a pruned weight w that serves as a good solution to Problem (8).

3.3. A multi-stage procedure

Our single-stage methods discussed in Section 3.2 lead to high-quality solutions for Problem (8). Compared to existing methods, for a given sparsity level, our algorithms deliver a better objective value for Problem (8)—for example, see Figure 2(b). However, we note that the final performance (e.g., accuracy) of the pruned network depends heavily on the quality of the local quadratic approximation. This is particularly true when targeting high levels of sparsity (i.e., zeroing out many weights), as the local objective function used in (8) may not accurately approximate the true loss function $\mathcal{L}$ globally. To this end, we propose a multi-stage procedure named CHITA++ that improves the approximation quality by iteratively updating and solving local quadratic models. We use a scheduler to gradually increase the sparsity constraint and take a small step towards higher sparsity in each stage to ensure the validity of the local quadratic approximation. Our multi-stage procedure leverages the efficiency of the single-stage approaches and can lead to pruned networks with improved accuracy by utilizing more accurate approximations of the true loss function. For example, our experiments show that the multi-stage procedure can prune ResNet20 to 90% sparsity in just a few minutes and increases test accuracy from 15% to 79% compared to the single-stage method. Algorithm 1 presents more details on CHITA++.

It is worth pointing out that our multi-stage algorithm is different from the gradual pruning approach described in Han et al. (2015). While both methods involve pruning steps, the gradual pruning approach also includes fine-tuning steps in which SGD is applied to further optimize the parameters for better results. However, these fine-tuning steps can be computationally expensive, usually taking days to run. In contrast, our proposed multi-stage method is a one-shot pruning method and only requires constructing and solving Problem (8) several times, resulting in an efficient and accurate solution. This solution can then be further fine-tuned using SGD or we can plug it into the gradual pruning framework, something we explore in Section 4.2.

Algorithm 1 CHITA++: a multi-stage pruning procedure

Require: Pre-trained weights $\bar{w}$, a target sparsity level τ, number of stages f.

1: **Initialization:** Construct a increasing sequence of sparsity parameters $\tau_1, \tau_2, \ldots, \tau_f = \tau$; and set $w^0 = \bar{w}$
2: **for** $t = 1, 2, \ldots, f$ **do**
3: At current solution w^{t-1}, calculate the gradient based on a batch of n data points and construct the matrix A given in (4).
4: Obtain a solution w^t to problem (8) by invoking CHITA$(A, \bar{w}, k)$ with $\bar{w} = w^{t-1}$ and number of nonzeros $k = \lfloor (1 - \tau_t)p \rfloor$.
5: **end for**

4. Experimental Results

We compare our proposed framework with existing approaches, for both one-shot and gradual pruning. Our code is publicly available at: https://github.com/mazumder-lab/CHITA

4.1. One shot pruning

We start by comparing the performance of our methods: CHITA (single-stage) and CHITA++ (multi-stage) with several existing state-of-the-art one-shot pruning techniques on various pre-trained networks. We use the same experimental setup as in recent work (Yu et al., 2022; Singh & Alistarh, 2020). The existing pruning methods we consider include MP (Magnitude Pruning, Mozer & Smolensky, 1989), WF (WoodFisher, Singh & Alistarh, 2020), CBS (Combinatorial Brain Surgeon, Yu et al., 2022) and M-FAC (Matrix-Free Approximate Curvature, Frantar et al., 2021). The pre-trained networks we use are MLPNet (30K parameters) trained on MNIST (LeCun et al., 1998), ResNet20 (He et al., 2016, 200k parameters) trained on CIFAR10 (Krizhevsky et al., 2009), and MobileNet (4.2M parameters) and ResNet50 (He et al., 2016, 22M parameters) trained on ImageNet (Deng et al., 2009). For further details on the choice of the Hessian approximation, we refer the reader to Appendix A.5. Detailed information on the experimental setup and reproducibility can be found in Appendix B.1.1.

4.1.1. Runtime comparison

Recent works that use the empirical Fisher information matrix for pruning purposes (Singh & Alistarh, 2020; Yu et al., 2022) show that using more samples for Hessian and gradient approximation results in better accuracy. Our experiments also support this conclusion. However, most prior approaches become computationally prohibitive as sample size n increases. As an example, the Woodfisher and CBS algorithms require hours to prune a MobileNet when n is set

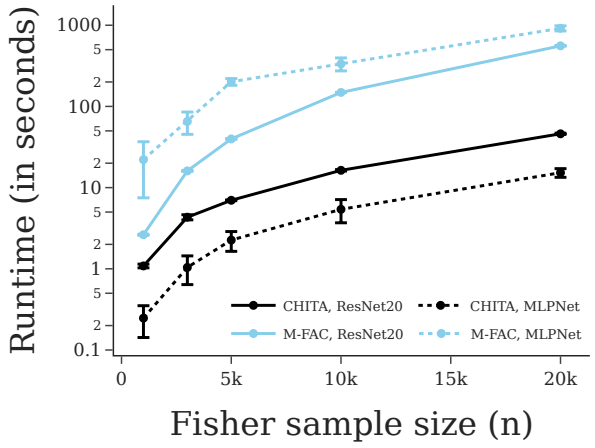


Figure 1: Runtime comparison between our single-stage approaches and M-FAC (the fastest among the competitive methods) while pruning MLPNet and ResNet20 to 90% sparsity level (90% of the entries are zero). Note that Woodfisher and CBS are at least 1000 times slower than M-FAC. The error bar represents the standard error over four runs. CHITA here uses IHT to find a support and performs a back-solve on the found support.

to 1000, and their processing time increases at least linearly with n. In contrast, our method has been designed with efficiency in mind, and we have compared it to M-FAC, a well-optimized version of Woodfisher that is at least 1000 times faster. The results, as depicted in Figure 1, demonstrate a marked improvement in speed for our algorithm, with up to 20 times faster performance.

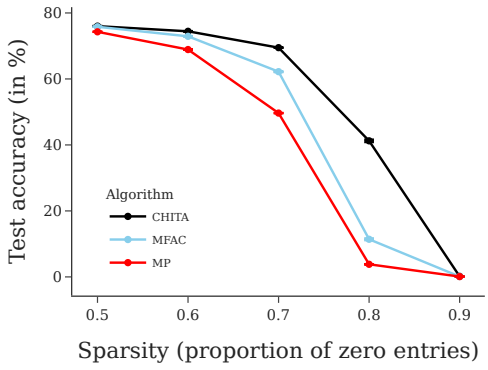
Network	Sparsity	MP	WF	CBS	CHITA	CHITA++
MLPNet on MNIST (93.97%)	0.5	93.93	94.02	93.96	93.97 (±0.03)	**95.97** (±0.05)
	0.6	93.78	93.82	93.96	93.94 (±0.02)	**95.93** (±0.04)
	0.7	93.62	93.77	93.98	93.80 (±0.01)	**95.89** (±0.06)
	0.8	92.89	93.57	93.90	93.59 (±0.03)	**95.80** (±0.03)
	0.9	90.30	91.69	93.14	92.46 (±0.04)	**95.55** (±0.03)
	0.95	83.64	85.54	88.92	88.09 (±0.24)	**94.70** (±0.06)
	0.98	32.25	38.26	55.45	46.25 (±0.85)	**90.73** (±0.11)
ResNet20 on CIFAR10 (91.36%)	0.3	90.77	**91.37**	91.35	**91.37 (±0.04)**	91.25 (±0.08)
	0.4	89.98	91.15	**91.21**	91.19 (±0.05)	**91.20** (±0.05)
	0.5	88.44	90.23	90.58	90.60 (±0.07)	**91.04** (±0.09)
	0.6	85.24	87.96	88.88	89.22 (±0.19)	**90.78** (±0.12)
	0.7	78.79	81.05	81.84	84.12 (±0.38)	**90.38** (±0.10)
	0.8	54.01	62.63	51.28	57.90 (±1.04)	**88.72** (±0.17)
	0.9	11.79	11.49	13.68	15.60 (±1.79)	**79.32** (±1.19)
MobileNetV1 on ImageNet (71.95%)	0.3	71.60	**71.88**	**71.88**	**71.87** (±0.01)	71.86 (±0.02)
	0.4	69.16	71.15	71.45	71.50 (±0.02)	**71.61** (±0.02)
	0.5	62.61	68.91	70.21	70.42 (±0.02)	**70.99** (±0.04)
	0.6	41.94	60.90	66.37	67.30 (±0.03)	**69.54** (±0.01)
	0.7	6.78	29.36	55.11	59.40 (±0.09)	**66.42** (±0.03)
	0.8	0.11	0.24	16.38	29.78 (±0.18)	**47.45** (±0.25)

Table 1: The pruning performance (model accuracy) of various methods on MLPNet, ResNet20, MobileNetV1. As to the performance of MP, WF, and CBS, we adopt the results reported in Yu et al. (2022). We take five runs for our single-stage (CHITA) and multi-stage (CHITA++) approaches and report the mean and standard error (in the brackets). The best accuracy values (significant) are highlighted in bold. Here sparsity denotes the fraction of zero weights in convolutional and dense layers.

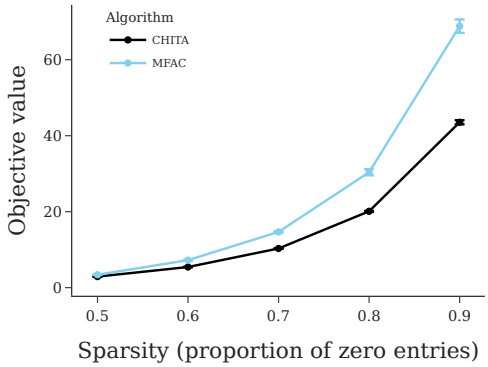
4.1.2. Accuracy of the Pruned Models

Comparison against state-of-the-art. Table 1 compares the test accuracy of MLPNet, ResNet20 and MobileNetV1 pruned to different sparsity levels. Our single-stage method achieves comparable results to other state-of-the-art approaches with much less time and memory consumption. The multi-stage method (CHITA++) outperforms other methods by a large margin, especially with a high sparsity rate (that is, with more aggressive pruning).

One-shot pruning on ResNet50. We further compare our approach to competing methods on ResNet50, an even larger network where some pruning algorithms, like CBS, do not scale. In Figure 2, we evaluate the performance of our algorithm in comparison to M-FAC and Magnitude Pruning (MP) using two metrics: test accuracy and the final objective value of the ℓ_0-constrained problem (8). As both M-FAC and our algorithm aim to minimize this objective, it can be used to judge the efficacy of our model in solving problem (8). As seen in the figure, our approach achieves a lower objective value, and in this case, it also results in a better test accuracy for the final pruned network.



(a) Test accuracy for one-shot pruning on ResNet50.



(b) The objective value in (8) for pruning ResNet50.

Figure 2: One-shot pruning on ResNet50 (Dense accuracy is 77.01%). The error bars are over four runs. For a fair comparison, both CHITA and M-FAC use the same hyperparameters and the same training samples for Hessian and gradient approximation.

Sparsity schedule in multi-stage procedure. We study the effect of the sparsity schedule that is, the choice of the sparsity parameters $\tau_1 \leq \tau_2 \leq \cdots \leq \tau_f = \tau$ in Algorithm 1 on the performance of CHITA++. We compare test accuracy of three different schedules: (i) exponential mesh, (ii) linear mesh, and (iii) constant mesh. For these schedules, f is set to be 15. For the first two meshes, τ_1 and τ_{15} are fixed as 0.2 and 0.9, respectively. As shown in Figure 3, the exponential mesh computes $\tau_2, \ldots, \tau_{14}$ by drawing an exponential function, while the linear mesh adopts linear interpolation (with τ_1 and τ_{15} as endpoints) to determine $\tau_2, \ldots, \tau_{14}$ and the constant mesh has $\tau_1 = \tau_2 = \cdots = \tau_{15}$.

Figure 4 plots the test accuracy of the three schedules over the number of stages. We observe that the linear mesh outperforms the exponential mesh in the first few iterations, but its performance drops dramatically in the last two iterations. It appears that in high sparsity levels, even a slight increase in the sparsity rate leads to a large drop in accuracy. Taking small "stepsizes" in high sparsity levels allows the exponential mesh to fine-tune the weights in the last several stages and achieve good performance.

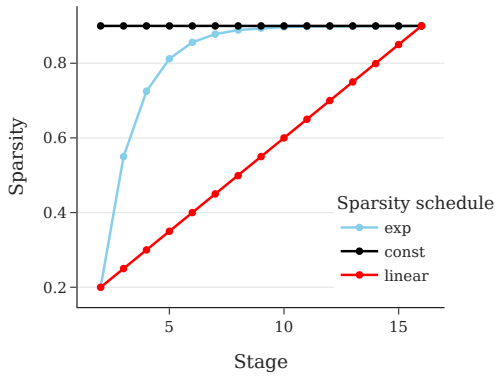


Figure 3: Three different sparsity schedules: exponential, linear, and constant schedules.

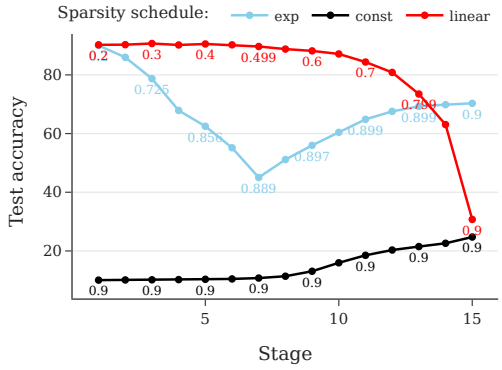


Figure 4: Comparison of test accuracy using CHITA++ with 15 stages, pruning a ResNet20 model to a 90% sparsity rate, under different sparsity schedules. Text around the point indicates the current sparsity level of the point.

Additional ablation studies. We perform additional ablation studies to further evaluate the performance of our method. These studies mainly focus on the effect of the ridge term (in Appendix B.2.1), and the effect of the first-order term (in Appendix B.2.2).

4.2. Performance of gradual pruning

To compare our one-shot pruning algorithms against more unstructured pruning methods, we use CHITA results into a gradual pruning procedure (Gale et al., 2019). We follow the approach in Singh & Alistarh (2020). Specifically, we alternate between pruning steps where a sparse weight is computed and fine-tuning steps on the current support via Stochastic Gradient Descent (SGD). To obtain consistent results, we start from the same pre-trained weights used in Kusupati et al. (2020), and re-train for 100 epochs using SGD during fine-tuning steps, similarly to Kusupati et al. (2020); Singh & Alistarh (2020). We compare our approach against Incremental (Zhu & Gupta, 2018), STR (Kusupati et al., 2020), Global Magnitude (Singh & Alistarh, 2020), WoodFisher (Singh & Alistarh, 2020), GMP (Gale et al., 2019), Variational Dropout (Molchanov et al., 2017), RIGL (Evci et al., 2020), SNFS (Dettmers & Zettlemoyer, 2020) and DNW (Wortsman et al., 2019). Further details on training procedure can be found in Appendix B.1.2.

Method	Sparsity (%)	Pruned Accuracy	Relative Drop (%) $\frac{Pruned-Dense}{Dense}$	Remaining # of params
Incremental	74.11	67.70	-4.11	1.09 M
STR	75.28	68.35	-5.07	1.04 M
Global Magnitude	75.28	69.90	-2.92	1.04 M
WoodFisher	75.28	70.09	-2.65	1.04 M
CHITA	75.28	**71.11**	**-1.23**	1.04 M
Incremental	89.03	61.80	-12.46	0.46 M
STR	89.01	62.10	-13.75	0.46 M
Global Magnitude	89.00	63.02	-12.47	0.46 M
WoodFisher	89.00	63.87	-11.29	0.46 M
CHITA	89.00	**67.68**	**-6.00**	0.46 M

Table 2: Results of gradually pruning MobilenetV1 in 75% and 89% sparsity regimes, comparing CHITA to other baselines (Dense accuracy: 72.00%). We also include the relative drop in accuracy to account for different methods starting from different dense weights. CHITA numbers are averaged across two runs. Numbers for other baselines are taken from Singh & Alistarh (2020).

MobileNetV1. We start by pruning MobileNetV1 (4.2M parameters). As Table 2 demonstrates, CHITA delivers significantly more accurate pruned models than previous state-of-the-art approaches at sparsity levels 75% and 89%. In particular, when pruning to a sparsity of 89%, there is only a 6% accuracy loss compared to the previous best result of 11.29%.

ResNet50. Similarly to MobileNetV1, CHITA improves test accuracy at sparsity levels 90%, 95%, and 98% compared to all other baselines, as Table 3 shows. This im-

provement becomes more noticeable as we increase the target sparsity, with CHITA producing a pruned model with 67.18% accuracy compared to 65.66%, the second-best performance, and previous state-of-the-art.

Method	Sparsity (%)	Pruned Accuracy	Relative Drop (%) $\frac{Pruned-Dense}{Dense}$	Remaining # of params
GMP + LS	90.00	73.91	-3.62	2.56 M
Variational Dropout	90.27	73.84	-3.72	2.49 M
RIGL + ERK	90.00	73.00	-4.94	2.56 M
SNFS + LS	90.00	72.90	-5.32	2.56 M
STR	90.23	74.31	-3.51	2.49 M
Global Magnitude	90.00	75.20	-2.42	2.56 M
DNW	90.00	74.00	-4.52	2.56 M
WoodFisher	90.00	75.21	-2.34	2.56 M
CHITA	90.00	**75.36**	**-2.14**	2.56 M
GMP	95.00	70.59	-7.95	1.28 M
Variational Dropout	94.92	69.41	-9.49	1.30 M
Variational Dropout	94.94	71.81	-6.36	1.30 M
RIGL + ERK	95.00	70.00	-8.85	1.28 M
DNW	95.00	68.30	-11.31	1.28 M
STR	94.80	70.97	-7.84	1.33 M
STR	95.03	70.40	-8.58	1.27 M
Global Magnitude	95.00	71.79	-6.78	1.28 M
WoodFisher	95.00	72.12	-6.35	1.28 M
CHITA	95.00	**73.13**	**-5.03**	1.28 M
GMP + LS	98.00	57.90	-24.50	0.51 M
Variational Dropout	98.57	64.52	-15.87	0.36 M
DNW	98.00	58.20	-24.42	0.51 M
STR	98.05	61.46	-20.19	0.50 M
STR	97.78	62.84	-18.40	0.57 M
Global Magnitude	98.00	64.28	-16.53	0.51 M
WoodFisher	98.00	65.55	-14.88	0.51 M
CHITA	98.00	**67.19**	**-12.75**	0.51M

Table 3: Results of gradually pruning a ResNet50 network in the 90%, 95%, and 98% sparsity regimes, comparing CHITA to other state-of-the-art methods (Dense accuracy: 77.01%). We also include the relative drop in accuracy to account for different methods starting from different dense weights. CHITA numbers are averaged across two runs. Numbers for other baselines are taken from Singh & Alistarh (2020).

5. Conclusion

In this work we have presented an efficient network pruning framework CHITA , which is based on a novel, Hessian-free ℓ_0-constrained regression formulation and combinatorial optimization techniques. Our single-stage methods demonstrate comparable results to existing methods while achieving a significant improvement in runtime and reducing memory usage. Furthermore, by building upon the single-stage methods, our multi-stage approach is capable of achieving even further improvements in model accuracy. Additionally, we have demonstrated that incorporating our pruning methods into existing gradual pruning frameworks results in sparse networks with state-of-the-art accuracy.

6. Acknowledgements

This research is supported in part by grants from the Office of Naval Research (N000142112841 and N000142212665) and Google. We acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing HPC resources that have contributed to the research results reported within this paper. Additionally, we thank Google for providing us with Google Cloud Credits to run some of the computational experiments reported in this paper. We thank Shibal Ibrahim for helpful discussions. We thank Thiago Serra and Yu Xin for sharing with us code from their CBS paper (Yu et al., 2022), and Srikumar Ramalingam for helpful clarifications on the computation cost of CBS.

References

Beck, A. and Eldar, Y. C. Sparsity constrained nonlinear optimization: Optimality conditions and algorithms. *SIAM Journal on Optimization*, 23(3):1480–1509, 2013.

Beck, A. and Teboulle, M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Bertsekas, D. P. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

Bertsimas, D. and Van Parys, B. Sparse high-dimensional regression: Exact scalable algorithms and phase transitions. *The Annals of Statistics*, 48(1):300–323, 2020.

Bertsimas, D., King, A., and Mazumder, R. Best subset selection via a modern optimization lens. *The annals of statistics*, 44(2):813–852, 2016.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., and Guttag, J. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

Blumensath, T. Accelerated iterative hard thresholding. *Signal Processing*, 92(3):752–756, 2012.

Blumensath, T. and Davies, M. E. Iterative hard thresholding for compressed sensing. *Applied and computational harmonic analysis*, 27(3):265–274, 2009.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Dettmers, T. and Zettlemoyer, L. Sparse networks from scratch: Faster training without losing performance, 2020. URL https://openreview.net/forum?id=ByeSYa4KPS.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dong, X., Chen, S., and Pan, S. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in Neural Information Processing Systems*, 30, 2017.

Evci, U., Elsen, E., Castro, P., and Gale, T. Rigging the lottery: Making all tickets winners, 2020. URL https://openreview.net/forum?id=ryg7vA4tPB.

Frantar, E., Kurtic, E., and Alistarh, D. M-fac: Efficient matrix-free approximations of second-order information. *Advances in Neural Information Processing Systems*, 34: 14873–14886, 2021.

Friedman, J., Hastie, T., and Tibshirani, R. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019. URL http://arxiv.org/abs/1902.09574.

Gordon, M. A., Duh, K., and Andrews, N. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*, 2020.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Hanson, S. and Pratt, L. Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems*, 1, 1988.

Hassibi, B. and Stork, D. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

Hazimeh, H. and Mazumder, R. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. *Operations Research*, 68(5):1517–1537, 2020.

Hazimeh, H., Mazumder, R., and Saab, A. Sparse regression at scale: Branch-and-bound rooted in first-order optimization. *Mathematical Programming*, 196(1-2):347–388, 2022.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Kusupati, A., Ramanujan, V., Somani, R., Wortsman, M., Jain, P., Kakade, S., and Farhadi, A. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Max, A. W. Inverting modified matrices. In *Memorandum Rept. 42, Statistical Research Group*, pp. 4. Princeton Univ., 1950.

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 2498–2507. JMLR.org, 2017.

Mozer, M. C. and Smolensky, P. Using relevance to reduce network size automatically. *Connection Science*, 1(1): 3–16, 1989.

Nesterov, Y. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

Nocedal, J. and Wright, S. J. *Numerical optimization*. Springer, 1999.

Singh, S. P. and Alistarh, D. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33: 18098–18109, 2020.

Thomas, V., Pedregosa, F., Merriënboer, B., Manzagol, P.-A., Bengio, Y., and Le Roux, N. On the interplay between noise and curvature and its effect on optimization and generalization. In *International Conference on Artificial Intelligence and Statistics*, pp. 3503–3513. PMLR, 2020.

Tillmann, A. M., Bienstock, D., Lodi, A., and Schwartz, A. Cardinality minimization, constraints, and regularization: a survey. *arXiv preprint arXiv:2106.09606*, 2021.

Wortsman, M., Farhadi, A., and Rastegari, M. *Discovering Neural Wirings*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Yao, Y., Rosasco, L., and Caponnetto, A. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

Yao, Z., Gholami, A., Keutzer, K., and Mahoney, M. Pyhessian: Neural networks through the lens of the hessian. In *ICML workshop on Beyond First-Order Optimization Methods in Machine Learning*, 2020.

Yu, X., Serra, T., Ramalingam, S., and 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.

Zhu, M. and Gupta, S. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Sy1iIDkPM.

A. Algorithmic details

A.1. IHT with aggressive stepsize

Challenges of stepsize choice Choosing an appropriate stepsize τ^s is crucial to achieving a faster convergence rate. In theory, setting τ^s as the constant $1/L$ in (9) is a common choice in the literature to ensure the convergence to a stationary solution (Bertsimas et al., 2016; Hazimeh & Mazumder, 2020), where L is the Lipschitz constant of the gradient of $Q(w)$. i.e., $\|\nabla Q(\alpha) - \nabla Q(\beta)\| \leq L\|\alpha - \beta\|$ for all α, $\beta \in \mathbb{R}^p$. Since Q is a quadratic objective, this quantity L is given by $L = n\lambda + \|A\|_2^2$, where $\|A\|_2$ is the maximum singular value of A. This quantity could be substantial when p is large, leading to very conservative updates, sometimes negligible. Moreover, the computation of L itself may involve a few power iterations or a randomized SVD algorithm, which could be as costly as several IHT updates. An alternative method for determining the stepsize is to use a backtracking line search, as proposed in Beck & Teboulle (2009). The method involves starting with a relatively large estimate of the stepsize and iteratively shrinking the step size until a sufficient decrease of the objective function is observed. However, this method requires multiple evaluations of the quadratic objective, which can be computationally expensive.

Our novel scheme We propose a new line search strategy to efficiently determine an aggressive stepsize to address the issue of slow updates in the IHT algorithm. Note that the problem of finding the best stepsize can be written as the following one-dimensional problem

$$\min_{\tau^s \geq 0} g(\tau^s) := Q\left(P_k\left(w^t - \tau^s \nabla Q(w^t)\right)\right). \tag{11}$$

Since P_k is a piecewise function, $g(\tau^s)$ is a univariate piecewise quadratic function which is generally non-convex, as illustrated in Figure 5. Our key observation is that the first breaking point of $g(\tau^s)$ and the optimal stepsize on the first piece can be computed easily. More specifically, denote by τ_c^s the first breaking point of $g(\tau^s)$. Namely, τ_c^s is the largest value of τ' such that the hard thresholding based on $\tau^s \in [0, \tau']$ does not change the support, i.e. $\mathcal{I}_k(w^t) = \mathcal{I}_k(w^t - \tau^s \nabla Q(w^t))$, $\forall \tau^s \in [0, \tau']$. Let us denote $\mathcal{S} := \text{supp}(w)$. In the case where $|\mathcal{S}| = k$, τ_c^s can be computed in closed form using

$$\tau_c^s = \min_{i \in \mathcal{S}} \left\{ \frac{|w_i^t|}{\max\{\max_{j \notin \mathcal{S}} |[\nabla Q(w^t)]_j| - [\nabla Q(w^t)]_i \text{sign}(w_i^t [\nabla Q(w^t)]_i), 0_+\}} \right\}. \tag{12}$$

As previously established, over the interval $\tau^s \in [0, \tau_c^s]$, the function $g(\tau^s) = Q\left(w^t - \tau^s \nabla Q(w^t)\right)$ is a quadratic function. Let us denote by τ_m^s the optimal value of τ^s that minimizes $g(\tau^s)$ within the interval $[0, \tau_c^s]$. It is straightforward to see that τ_m^s can be computed in closed form with the same computational cost as a single evaluation of the quadratic objective function.

If $\tau_m^s < \tau_c^s$, then the optimal value of τ^s lies within the first quadratic piece. Practically, we have found that in this case τ_m^s is often also the global minimum of $g(\tau^s)$. Therefore, we can confidently take the stepsize as $\tau^s = \tau_m^s$. Otherwise if $\tau_m^s = \tau_c^s$, then we know that $g(\tau^s)$ is monotonically decreasing on the interval $[0, \tau_c^s]$. This implies that $g(\tau^s)$ would likely continue to decrease as τ^s becomes larger than τ_c^s. As a result, we perform a line search by incrementally increasing the value of τ^s by a factor of $\gamma > 1$ starting from τ_c^s to approximate the stepsize that results in the steepest descent. The above procedure is summarized in Algorithm 2.

Our proposed scheme offers a significant improvement in efficiency compared to standard backtracking line search by eliminating redundant steps on the quadratic piece of $g(\tau^s)$ over $[0, \tau_c^s]$. Additionally, our method directly computes the optimal stepsize on the first piece, which in many cases, results in a greater reduction in the objective function when compared to the standard backtracking line search.

Finally, we note that during line search, it is always possible to find the piece of the quadratic function to which the current stepsize τ^s belongs, say $[\tau_l^s, \tau_u^s]$, and calculate the optimal stepsize over that piece with small extra costs to further improve the line search. But we find it unnecessary in practice as the line search procedure usually terminates in a few steps.

A.2. Cyclic coordinate descent

Although IHT does well in identifying and updating the support, we observe that it makes slow progress in decreasing the objective in experiments. To address this issue, we use cyclic coordinate descent (CD, Bertsekas, 1997; Nesterov, 2012)

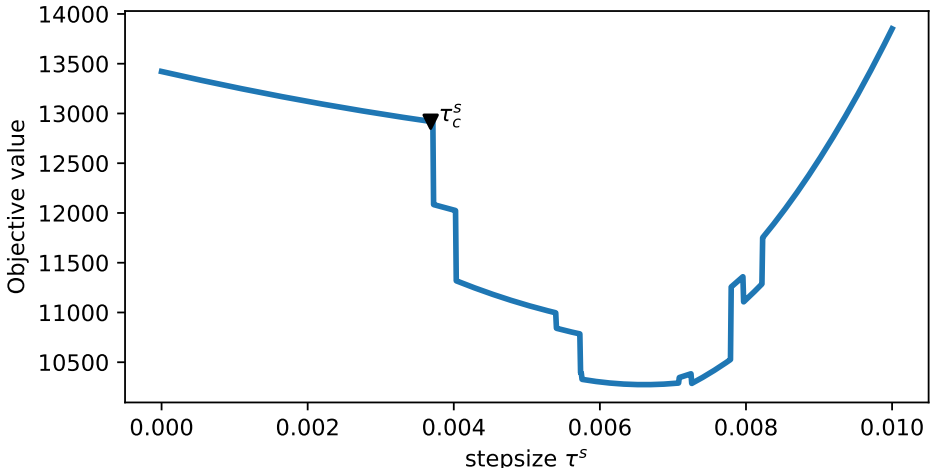


Figure 5: An example of $g(\tau^s)$ for $n = 100, p = 200$ and $k = 10$. The black triangle denotes the first breaking point of $g(\tau^s)$, with its x-coordinate represented by τ_c^s.

Algorithm 2 A novel scheme to determine the stepsize τ^s

Require: $w^t, k, \gamma > 1$

Compute τ_c^s as the first breaking point of $g(\tau^s)$ in closed form. {Time complexity $O(p)$}

Compute

$$\tau_m^s = \operatorname*{arg\,min}_{\tau^s \in [0, \tau_c^s]} g(\tau^s) := Q\left(P_k\left(w^t - \tau^s \nabla Q(w^t)\right)\right) \tag{13}$$

as the optimal stepsize on the first piece of $g(\tau^s)$ in closed form. {Time complexity $O(nk)$}

if $\tau_m^s < \tau_c^s$ **then**
  $\tau^s \leftarrow \tau_m^s$.
else
  $g_{\text{best}} \leftarrow g(\tau_c^s)$, and $\tau^s \leftarrow \tau_c^s$.
  while $g_{\text{best}} > g(\gamma\tau^s)$ **do**
    $g_{\text{best}} \leftarrow g(\gamma\tau^s)$, and $\tau^s \leftarrow \gamma\tau^s$.
  end while
end if

with full minimization in every nonzero coordinate to refine the solution on the support. CD-type methods are widely used for solving huge-scale optimization problems in statistical learning, especially those problems with sparsity structure, due to their inexpensive iteration updates and capability of exploiting problem structure, such as Lasso (Friedman et al., 2010) and L_0L_2-penalized regression (Hazimeh & Mazumder, 2020).

Our cyclic CD updates each nonzero coordinate (with others fixed) as per a cyclic rule, and skips any coordinate with zero value to avoid violating the ℓ_0 constraint. With a feasible initialization w^t and a coordinate i in the support of w^t, w_i^{t+1} is obtained by optimizing the i-th coordinate (with others fixed) through:

$$w_i^{t+1} = \mathsf{CDUpdate}(w^t, i) := \operatorname*{arg\,min}_{w \in \mathbb{R}} Q\left(w_1^t, \ldots, w_{i-1}^t, w, w_{i+1}^t, \ldots, w_p^t\right). \tag{14}$$

Calculating $\mathsf{CDUpdate}(w^t, i)$ requires the minimization of a univariate quadratic function with time cost $O(n)$.

Cyclic CD enjoys a fast convergence rate (Bertsekas, 1997; Nesterov, 2012). However, the quality of the resulting solution is limited and depends heavily on the initial solution, as CD cannot modify the support of a solution. In practice, we adopt a hybrid updating rule that combines IHT and cyclic CD for better performance in terms of both quality and efficiency. In each iteration, we perform several rounds of IHT updates and then apply cyclic CD to refine the solution on the support. This approach is summarized in Algorithm 3.

Algorithm 3 IHT with CD: IHT-CD$(w^0, k, t_{\mathsf{HT}}, t_{\mathsf{CD}})$

Require: w^0, k, $t_{\mathsf{HT}}, t_{\mathsf{CD}}$
for $t = 0, 1, \ldots$ **do**
  $w \leftarrow w^t$
  for $t' = 1, \ldots, t_{\mathsf{HT}}$ **do**
    Compute stepsize τ^s using Algorithm 2.
    $w \leftarrow \mathsf{HT}(w, k, \tau^s)$ {Time complexity $O(np)$}
  end for
  for $t' = 1, \ldots, t_{\mathsf{CD}}$ **do**
    for $i \in \mathrm{supp}(w)$ **do**
      $w \leftarrow \mathsf{CDUpdate}(w, i)$ {Time complexity $O(n)$}
    end for
  end for
  $w^{t+1} \leftarrow w$
end for

A.3. Active set updates

The active set strategy is a popular approach that has been shown to be effective in reducing complexity in various contexts (Nocedal & Wright, 1999; Friedman et al., 2010; Hazimeh & Mazumder, 2020). In our problem setting, the active set strategy works by starting with an initial active set (of length equal to a multiple of the required number of nonzeros k, e.g., $2k$) that is selected based on the magnitude of the initial solution. In each iteration, we restrict the updates of Algorithm 3 to the current active set $\mathcal{A}$. After convergence, we perform IHT updates on the full vector to find a better solution w with $\mathrm{supp}(w) \not\subseteq \mathcal{A}$. The algorithm terminates if such w does not exist; otherwise, we update $\mathcal{A} \leftarrow \mathcal{A} \cup \mathrm{supp}(w)$, and the process is repeated. Algorithm 4 gives a detailed illustration of the active set method, with Algorithm 3 as the inner solver (potentially the inner solver can be replaced with any other solver, such as Algorithm 5 in the next section). In our experiments, this strategy works well on medium-sized problems ($p \sim 10^5$) and sparse problems ($k \ll p$).

Algorithm 4 Active set with IHT: CHITA-CD$(w^0, k, t_{\mathsf{HT}}, t_{\mathsf{CD}}, \mathcal{A}^0)$

Require: w^0, k, $t_{\mathsf{HT}}, t_{\mathsf{CD}}$, and an initial active set $\mathcal{A}^0$
for $t = 0, 1, \ldots$ **do**
  $w^{t+1/2}|_{\mathcal{A}^t} \leftarrow$ IHT-CD$(w^t|_{\mathcal{A}^t}, k, t_{\mathsf{HT}}, t_{\mathsf{CD}})$ restricted on $\mathcal{A}^t$
  Find τ^s via line search such that $w^{t+1} \leftarrow \mathsf{HT}(w^{t+1/2}, k, \tau^s)$ satisfies
    (i) $Q(w^{t+1}) < Q(w^{t+1/2})$ (ii) $\mathrm{supp}(w^{t+1}) \not\subseteq \mathcal{A}^t$
  if such τ^s does not exist **then**
    break
  else
    $\mathcal{A}^{t+1} \leftarrow \mathcal{A}^t \cup \mathrm{supp}(w^{t+1})$
  end if
end for

A.4. Backsolve via Woodbury formula

As the dimensionality of the problem increases, CHITA-CD becomes increasingly computationally expensive. To address this issue, we propose a backsolve approach that further reduces the complexity while maintaining a slightly suboptimal solution. The backsolve approach calculates the optimal solution exactly on a restricted set. We first apply IHT updates a few times to obtain an initial feasible solution w, and then restrict the problem to the set $\mathcal{S} := \mathrm{supp}(w)$. Under the restriction, problem (8) reduces to a quadratic problem without ℓ_0 constraint and its minimizer reads

$$w^*_{\mathcal{S}} = (n\lambda I_k + A_{\mathcal{S}}^\top A_{\mathcal{S}})^{-1}(n\lambda \bar{w}_{\mathcal{S}} + A_{\mathcal{S}}^\top b), \tag{15}$$

where $A_{\mathcal{S}} \in \mathbb{R}^{n\times k}$ denotes a submatrix of A with columns only in $\mathcal{S}$. By exploiting the low-rank structure of $A_{\mathcal{S}}^\top A_{\mathcal{S}}$ and utilizing Woodbury formula (Max, 1950), (15) can be computed in $O(n^2 k)$ operations. Specifically, one can compute (15)

using matrix-vector multiplications involving only $A_{\mathcal{S}}$ (or its transpose) and one matrix-matrix multiplication via

$$\begin{aligned} w^*_{\mathcal{S}} &= (n\lambda)^{-1}[I_k - A_{\mathcal{S}}^\top (n\lambda I_k + A_{\mathcal{S}} A_{\mathcal{S}}^\top)^{-1} A_{\mathcal{S}}] \cdot (n\lambda \bar{w}_{\mathcal{S}} + A_{\mathcal{S}}^\top b) \\ &= \bar{w}_{\mathcal{S}} + (n\lambda)^{-1} A_{\mathcal{S}}^\top b - (n\lambda)^{-1} A_{\mathcal{S}}^\top (n\lambda I_k + A_{\mathcal{S}} A_{\mathcal{S}}^\top)^{-1} A_{\mathcal{S}} (n\lambda \bar{w}_{\mathcal{S}} + A_{\mathcal{S}}^\top b). \end{aligned} \tag{16}$$

The backsolve method is stated in Algorithm 5.

Algorithm 5 Backsolve: CHITA-BSO$(w^0, k, t_{\mathsf{HT}})$

Require: w^0, k, t_{HT}.
1: Construct an initial active set $\mathcal{A}^0$
2: $w \leftarrow$ CHITA-CD$(w^0, k, t_{\mathsf{HT}}, 0, \mathcal{A}^0)$
3: $\mathcal{S} \leftarrow \operatorname{supp}(w)$
4: $w_{\mathcal{S}} \leftarrow (n\lambda I_k + A_{\mathcal{S}}^\top A_{\mathcal{S}})^{-1}(n\lambda \bar{w}_{\mathcal{S}} + A_{\mathcal{S}}^\top b_{\mathcal{S}})$, using (16) {Time complexity $O(n^2 k)$}

We note that prior works (Singh & Alistarh, 2020; Yu et al., 2022; Hassibi & Stork, 1992) also use the formula, but they do not exploit the problem structure to reduce the runtime and memory consumption.

A.5. Stratified block-wise approximation

We describe in this subsection a block approximation strategy whereby we only consider limited-size blocks on the diagonal of the Hessian matrix and ignore off-diagonal parts. Given a disjoint partition $\{B_i\}_{i=1}^c$ of $\{1, 2, \ldots, p\}$ and assume blocks of size $B_1 \times B_1, \ldots, B_c \times B_c$ along the diagonal, problem (8) can then be decomposed into the following subproblems $(1 \le i \le c)$

$$\min_{w \in \mathbb{R}^{|B_i|}} \frac{1}{2}\|b_i - A_{B_i} w\|^2 + \frac{n\lambda}{2}\|w - \bar{w}_{B_i}\|^2, \ \text{ s.t. } \|w\|_0 \le k_i, \tag{17}$$

where $b_i = A_{B_i}\bar{w}_{B_i} - e$ and $\sum_{i=1}^c k_i = k$ determines the sparsity in each block. The difference in the selection of $\{k_i\}_{i=1}^c$ will greatly affect the quality of the solution. We observe in experiments that the best selection strategy is to first apply magnitude pruning (or other efficient heuristics) to get a feasible solution w, and then set $k_i = |\operatorname{supp}(w) \cap B_i|$, $\forall 1 \le i \le c$. Algorithm 6 states the block-wise approximation algorithm, with Algorithm 5 as the inner solver for each subproblem.

In our experiment, we adopt the same strategy to employ the block-wise approximation as in the prior work (Yu et al., 2022; Singh & Alistarh, 2020). We regard the set of variables that corresponds to a single layer in the network as a block and then subdivide these blocks uniformly such that the size of each block does not exceed a given parameter $B_{size} = 10^4$.

We clarify that the introduction of block-wise approximation is for the sake of solution quality (accuracy of pruned network) rather than algorithmic efficiency. This differs from previous works (Singh & Alistarh, 2020; Yu et al., 2022). In fact, solving (17) for $i = 1, \ldots, c$ requires operations of the same order as solving (8) directly. On the other side, we observe in our experiments that adopting block-wise approximation will dramatically increase the network MobileNet's accuracy (from 0.2% to near 30%, given a sparsity level of 0.8).

Algorithm 6 CHIAT-BSO with block approximation

Require: w^0, k, t_{HT}, a disjoint partition $\{B_i\}_{i=1}^c$ of $\{1, 2, \ldots, p\}$.
1: Obtain a feasible solution via magnitude pruning $w \leftarrow P_k(w^0)$.
2: **for** $i = 1, 2, \ldots, c$ **do**
3: Determine sparsity level $k_i = |\operatorname{supp}(w) \cap B_i|$
4: $w_{B_i} \leftarrow$ CHITA-BSO$(w_{B_i}, k_i, t_{\mathsf{HT}})$
5: **end for**

B. Experiment details

B.1. Experimental setup

B.1.1. One-shot pruning experiments

All experiments were carried on a computing cluster. Experiments for MLPNet and ResNet20 were run on an Intel Xeon Platinum 8260 machine with 20 CPUs; experiments for MobileNetV1 and ResNet50 were run on an Intel Xeon Gold 6248 machine with 40 CPUs and one GPU.

Algorithmic setting We utilize the CHITA algorithm with active set strategy and coordinate descent as acceleration techniques, as outlined in Algorithm 4, to prune MLPNet and ResNet20 networks. Additionally, we use Algorithm 4 as the inner solver of our proposed multi-stage approach, CHITA++, for these networks. As to MobileNetV1 and ResNet50, we utilize CHITA-BSO with block approximation (Algorithm 6) for solving single-stage problems. We employ the exact block-wise approximation strategy as applied in previous work (Yu et al., 2022; Singh & Alistarh, 2020), see Section A.5 for details. We also use Algorithm 6 as the inner solver of our proposed multi-stage procedure CHITA++ for these networks. In our experiments, we set the number of stages in CHITA++ to 15 for MLPNet and ResNet20 and 100 for MobileNetV1. CHITA++ results on MobileNetV1 are averaged over 4 runs.

Hyper-parameters For each network and each sparsity level, we run our proposed methods CHITA (single-stage) and CHITA++ (multi-stage) with ridge value λ ranging from $[10^{-5}, 10^{3}]$ and the number of IHT iterations (if Algorithm 4 is applied) ranging from $[5, 500]$. In single-stage settings, we consider solving problem (8) with/without the first-order term. We report in Table 1 the best model accuracy over all possible hyper-parameter combinations.

Hyper-parameters for ResNet50 experiments To obtain consistent results, we run CHITA and M-FAC with the same set of hyperparameters ($\lambda = 10^{-5}, n = 500, B_{size} = 10^{4}$) and on the same training samples for Hessian and gradient approximation. We performed a sensitivity analysis with different block sizes B_{size} and found similar results — suggesting that the results are robust to the choice of B_{size}.

Fisher sample size and mini-batch size In practice, we replace each $\nabla \ell_i(\bar{w})$ used in Hessian and gradient approximation by the average gradient of a mini-batch of size m. We display in Table 4 the Fisher sample size n and the mini-batch size m (also called fisher batch size) used for gradient evaluation. Note that WoodFisher, CBS, and CHITA utilize the same amount of data samples and the same batch size for MLPNet, ResNet20, and ResNet50; while for MobileNetV1, CHITA performs gradient evaluations on 16,000 training samples, which is much less compared to WF and CBS as they require 960,000 samples.

Model	MLPNet		ResNet20		MobileNetV1		ResNet50	
	sample	batch	sample	batch	sample	batch	sample	batch
CHITA	1000	1	1000	1	1000	16	500	16
WF & CBS	1000	1	1000	1	400	2400	-	-
M-FAC	-	-	-	-	-	-	500	16

Table 4: Comparisons of the Fisher sample size n and the mini-batch size m used in Hessian and gradient approximation on MLPNet, ResNet20, MobileNetV1 and ResNet50.

B.1.2. Gradual pruning

All experiments were carried on a computing cluster. Experiments for MobileNetV1 were run on an Intel Xeon Platinum 6248 machine with 30 CPUs and 2 GPUs; experiments for ResNet50 were run on five Intel Xeon Platinum 6248 machines with 200 CPUs and 10 GPUs.

Details on the pruning step In all our gradual pruning experiments, we begin by pruning the networks to a sparsity level of 50% and proceed with six additional pruning steps to reach the target sparsity. We follow the polynomial schedule

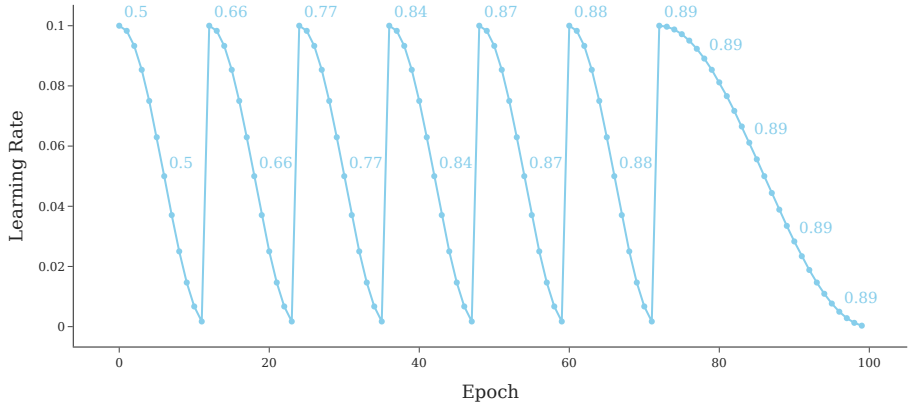


Figure 6: Learning rate schedule used in our gradual pruning experiments on MobileNetV1, with a target sparsity level of 0.89. Text around a point indicates the sparsity of the network at the current epoch.

introduced by Zhu & Gupta (2018) as the pruning schedule and use the CHITA-BSO algorithm with block approximation (Algorithm 6) as the pruning method. The block size is set to $B_{size} = 2000$ for MobileNetV1 and $B_{size} = 500$ for ResNet50.

Details on the fine-tuning process We incorporate SGD with a momentum of 0.9 for 12 epochs between pruning steps. Once the networks have been pruned to the target sparsity, we continue to fine-tune the networks for an additional 28 epochs using SGD with a momentum of 0.9 (total of 100 epochs). We utilize distributed training and set the batch size to 256 per GPU during the SGD training process.

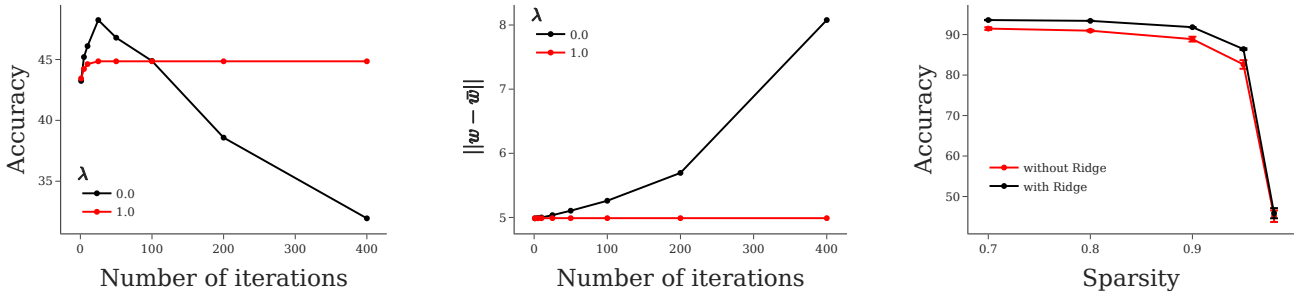
We implement a cosine-based learning rate schedule similar to the one used in the STR method (Kusupati et al., 2020). Specifically, the learning rate for each epoch e between two pruning steps that occur at epochs e_1 and e_2 is defined as:

$$0.00001 + 0.5 \times (0.1 - 0.00001) \times \left(1 + \cos\left(\pi \frac{e - e_1}{e_2 - e_1}\right)\right) \tag{18}$$

Figure 6 illustrates how such a learning rate schedule decays between pruning steps.

B.2. Implementation details and ablation studies

B.2.1. Effect of the Ridge term



(a) Test accuracy with respect to the number of iterations in a single run of CHITA-CD (Algorithm 4). The sparsity level is 0.98.

(b) The distance $||w - \bar{w}||$ with respect to the number of iterations in a single run of CHITA-CD (Algorithm 4). The sparsity level is 0.98.

(c) Test accuracy at different sparsity levels, with best ridge term selected in range $[10^{-5}, 1]$. The result is averaged over 4 runs.

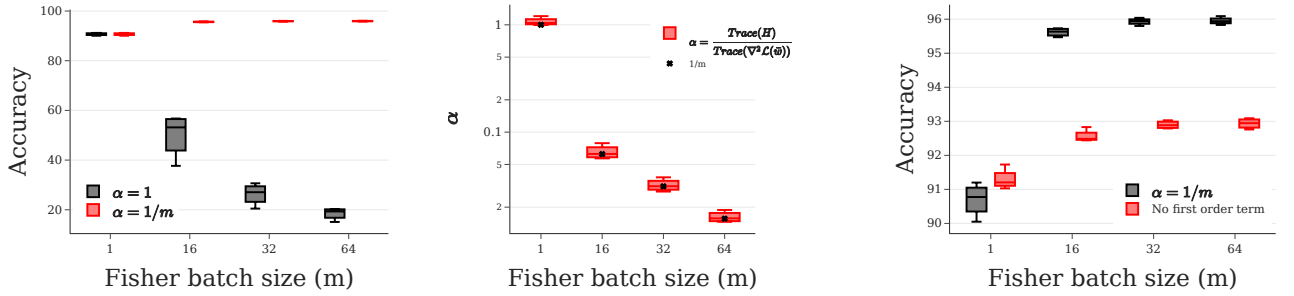
Figure 7: Effect of the ridge term on the test accuracy and pruned weights of MLPNet.

In this section, we study the effect of the ridge term on the performance of our algorithm, specifically focusing on the test accuracy over the course of the algorithm. As depicted in Figure 7(a), when no ridge term is applied, the test accuracy

increases initially but then experiences a sharp decline as the algorithm progresses. The underlying cause is revealed in Figure 7(b), which illustrates that without the ridge term, the distance between the original weight $\bar{w}$ and the pruned weight w keeps increasing as the algorithm progresses. As this distance increases, the local quadratic model used in (8) becomes less accurate, leading to poor test performance.

One solution to this problem would be to employ early stopping to prevent the distance from growing too large. However, determining the optimal stopping point can be challenging. Practically, we instead add the ridge term $\frac{n\lambda}{2}\|w-\bar{w}\|^2$ to the objective function, effectively regularizing the model and maintaining its accuracy. As shown in Figure 7(c), utilizing a well-tuned ridge term results in an increase of approximately 3% on MLPNet.

B.2.2. Effect of the first-order term



(a) Test accuracy with respect to mini-batch size, using different scaling factors.

(b) The difference between α calculated by (20) and our proposed heuristic $1/m$.

(c) Test accuracy with respect to mini-batch size, with and without the first-order term

Figure 8: Effect of using a scaled first-order term on pruning MLPNet with our proposed multi-stage solver CHITA++ to a sparsity level of 0.95. All results are averaged over 5 runs.

Mini-batches are used for gradient evaluations in practice instead of evaluating gradient $\nabla\ell_i(\bar{w})_{i=1}^n$ on n training samples. This means that each $\nabla\ell_i(\bar{w})$ is replaced by the average gradient of a mini-batch of size m. In this scenario, the empirical Fisher matrix H is not an accurate representation of the true Hessian matrix. However, it still provides a reasonable approximation but with a scaling factor (Thomas et al., 2020; Singh & Alistarh, 2020).

In scale-independent applications, e.g., minimizing $\mathcal{L}(w) \approx \mathcal{L}(\bar{w}) + \frac{1}{2}(w-\bar{w})^\top H(w-\bar{w})$ as considered in Singh & Alistarh (2020) and Yu et al. (2022), the empirical Fisher matrix H still effectively approximates the true Hessian. However, this approximation is no longer accurate in our framework, which includes a first-order term. This is supported by the results shown in Figure 8(a), where our framework with a correctly scaled term ($\alpha = 1/m$) demonstrates significantly improved performance compared to one without a scaling factor ($\alpha = 1$), especially when the fisher batch size m is much greater than one.

To address this issue, we propose a local quadratic approximation with a scaled first-order term that reads

$$Q(w) = \mathcal{L}(\bar{w}) + \alpha g^\top(w-\bar{w}) + \frac{1}{2}(w-\bar{w})^\top H(w-\bar{w}). \tag{19}$$

Our proposed ℓ_0-constrained framework (8) can be generalized to solving this problem by setting $y = A\bar{w} - \alpha e$, where e is a vector of ones. We propose an accurate estimation of α as

$$\alpha = \frac{\text{Trace}(H)}{\text{Trace}(\nabla^2\mathcal{L}(\bar{w}))}. \tag{20}$$

However, the computation cost of $\text{Trace}(\nabla^2\mathcal{L}(\bar{w}))$ is not negligible, even using accelerated methods as proposed in Yao et al. (2020). Through experimentation, as shown in Figure 8(b), we have discovered that the estimated value of α as given by (20) is relatively close to $1/m$. Therefore, we have chosen to use $1/m$ as a heuristic scaling factor in our experiments, as it provides a good approximation while reducing the computational cost.

In Figure 8(c), we further illustrate the benefits of using large mini-batches and a scaled first-order term. As the fisher batch size m increases, we can construct more precise local quadratic approximations through better estimation of H and g,

resulting in improved test accuracy. Additionally, when m is greater than 1, using a correctly scaled first-order term provides an additional performance boost.