OPT2022: 14th Annual Workshop on Optimization for Machine Learning

Network Pruning at Scale: A Discrete Optimization Approach

Wenyu Chen WENYU@MIT.EDU
Riade Benbaki RBENBAKI@MIT.EDU
Xiang Meng MENGX@MIT.EDU
Rahul Mazumder RAHULMAZ@MIT.EDU
Massachusetts Institute of Technology, Cambridge, MA

Abstract

Due to the ever-growing size of neural network models, there has been an emerging interest in compressing (i.e., pruning) neural networks by sparsifying weights in a pre-trained neural network, while maintaining the performance of the dense model as much as possible. In this work, we focus on a neural network pruning framework based on local quadratic models of the loss function. We present an optimization-based approach with an ℓ_0-regression formulation, and propose novel algorithms to obtain good solutions to the combinatorial optimization problem. In practice, our basic (single-stage) approach, based on one local quadratic model approximation, is up to 10^3 times faster than existing methods while achieving similar accuracy. We also propose a multi-stage procedure that outperforms other methods in terms of accuracy for a given sparsity constraint while remaining computationally efficient. In particular, our approach results in a 98% sparse (i.e., 98% of weights in the dense model are set to zero) MLPNet with 90% test accuracy (i.e., 3% reduction in test accuracy compared to the dense model)—this is an improvement over the previous best accuracy (55%).

1. Introduction

In the past decades, there have been great advancements in deep learning models [24, 29]. Many neural network models that show excellent accuracy metrics have a huge number of parameters (e.g. [10, 18])—resulting in increased costs during both the development and the deployment phases. One approach to mitigate costs at the deployment phase is to prune or sparsify networks by removing some parameters from a large pre-trained model—the goal being to obtain a network with similar performance but with a smaller number of parameters. This makes storage and deployment cheaper and easier, especially in resource-constrained environments. Pruning might also improve the robustness of network models towards adversarial manipulation [32].

Related work Depending on how weights are pruned, pruning methods can be categorized as magnitude-based pruning methods (e.g. [14, 15, 27]); impact-based pruning methods (e.g. [16, 22, 30]); optimization-based pruning methods (e,g. [1, 19, 35]), etc. The paper [6] provides a nice survey of these approaches; see also [35] and references therein. In this paper, we focus on an optimization based approach following a line of work that prunes the network based on local quadratic models of the loss function $\mathcal{L}(w)$. This is obtained via the second-order Taylor expansion around the pre-trained weights $\bar{w}$ of the neural network, i.e.

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

This line of work dates back to Optimal Brain Damage (OBD) by [22] and Optimal Brain Surgeon (OBS) by [16], and has been recently studied in a series of interesting works [12, 30, 35].

Studies on this topic that precede the work of [35] consider pruning network weights with the smallest (marginal) impact on the quadratic model, and potentially update the other weights. Different ways to approximate the Hessian matrix $\nabla^2 \mathcal{L}(\bar{w})$ have been proposed, based on various assumptions. OBD assumes the Hessian matrix to be diagonal and computes it efficiently by one back propagation. This approach however does not account for possible interactions across different weights. OBS extends this by assuming a general form for the Hessian matrix, and proposing the use of the empirical Fisher matrix as an approximation to the Hessian. As networks become larger, handling the (inverse) Hessian matrix becomes prohibitively difficult as its size scales with the square of the dimension. Several approaches have been attempted to make this more manageable. [11] proposes Layerwise OBS where pruning is done independently across each layer. WoodFisher [30] assumes a block diagonal structure for the Hessian, and proposes to estimate it by the empirical Fisher matrix, using only a subsample of all observations. They are able to prune neural networks jointly up to 25M parameters. M-FAC [12] further improves the implementation of WoodFisher, which can scale the pruning up to 25M without a block diagonal assumption.

All aforementioned approaches [11, 12, 16, 22, 30] prune weights based on their marginal impact on the local quadratic model—this pruning is not based on a joint optimization criterion. It appears that these approaches were used as minimizing the local quadratic model was considered to be too expensive to be practical. Interestingly, a recent method CBS (Combinatorial Brain Surgeon [35]) formulates the pruning problem as a mixed-integer quadratic program (MIQP, [21]) and proposes heuristics to obtain good solutions to the problem. This approach results in significant improvements in terms of pruning performance by considering the joint effects of pruning multiple weights simultaneously. However, the current CBS approach, which uses the same Hessian approximation as WoodFisher, requires the storage and use of the full $p \times p$ (inverse) Hessian matrix, p being the number of trainable parameters in the model. This limits the scalibility of the approach, in terms of both the computational time and memory utilization, as p is often in the order of millions and more (corresponding to a Hessian size $p^2 \gtrsim 10^{12}$).

Our approach In this work, we make use of a simple but important observation with which we can avoid computing and storing the Hessian matrix to solve the MIQP problem. We propose a reformulation of the problem into an ℓ_0-constrained sparse linear regression problem with a data matrix $X \in \mathbb{R}^{n \times p}$, where $n \lesssim 10^3$ is the number of the sub-samples used in the empirical Fisher information to approximate the Hessian. Compared to other state-of the-art approaches that consider the full Hessian, our approach leads to a significant reduction in memory usage (up to 10^3 times) without any approximation—this allows us to tackle large problems without resorting to block-diagonal approximations (employed by earlier approaches to reduce memory usage).

We also propose to include a first-order term in the local quadratic approximation and a small ridge regularization on $w - \bar{w}$, as we see gains from including these terms under different circumstances. Furthermore, we propose a novel approach to minimize our ℓ_0 regression reformulation, leveraging active set strategies, an aggressive choice of step size, and various methods to polish weights on the selected support. Our proposed approach leads to significant improvements over iterative hard thresholding methods (IHT, [7]) commonly used in sparse learning literature. Our framework can prune a network with 4.2M parameters to 80% sparsity (i.e., 0.84M nonzero parameters) in a minute.

Since the local quadratic model approximates the loss function only in a small neighbourhood (or trust-region) of the current solution [30], we propose a multi-stage algorithm along with a sparsity scheduler (from dense to sparse) where we update the local quadratic model during pruning (but without retraining) and solve a more constrained problem in each stage. Our experiments show that the resulting pruned models have a notably bigger accuracy compared to that from our single-stage algorithm and other pruning approaches.

The optimization algorithms proposed here are motivated by a recent body of work on combinatorial optimization-based methods for sparse linear regression [4, 17, 26]. The problem we consider here and our approach is different in several aspects. [17] focus on the unconstrained $\ell_0\ell_2$-regularized regression problem as opposed to the ℓ_0 constrained problem arising in the network pruning problem. Second, the problem size p we consider in this work is in the magnitude of millions, much larger than what is usually considered in prior works. In this work, to maintain sufficiently accurate pruned models, we consider solutions that are much denser, with a few hundreds of thousands of nonzeros. Finally, in the sparse learning literature in linear models, the optimization process usually starts from a trivial solution with all zeros and gradually increases the number of nonzeros through warm-start continuation. Here, on the contrary, we start from the current (dense) pre-trained weights and apply a dense-to-sparse cardinality scheduler to gradually remove more weights in our multi-stage algorithm.

Contributions Our contributions can be summarized as follows:

(a) We present a new optimization framework for network pruning based on a local quadratic approximation. In contrast to earlier approaches, we use a first-order term and an extra ridge term, which leads to improvements compared to other approaches without them.

(b) We reformulate the optimization problem into an ℓ_0-constrained linear regression problem: We ameliorate the pitfalls of storing a large dense Hessian as employed by current approaches, resulting in a $10^2 \sim 10^3$ times smaller memory usage

(c) We propose novel algorithms to obtain good solutions to the sparse regression reformulation, which leads to up to 1000 times runtime improvement compared to existing network pruning algorithms

(d) Making use of (c), we propose a multi-stage approach with a dense-to-sparse sparsity scheduling. Applying this approach to MLPNet yields a 98% sparse network with 90% test accuracy (3% less than the dense one), which is a significant improvement over previous results.

Organization The rest of paper is organized as follows: In Section 2, we describe the problem setup of network pruning considered in this paper and state our optimization formulation. We present in Section 3 our algorithmic framework for solving the network pruning problem, including both single-stage and multi-stage approaches. In Section 4, we demonstrate the usefulness of our framework in terms of computational time, memory usage and model performance, and conduct some ablation studies. Additional algorithm details and experimental details are provided in Appendix A and B.

2. Problem Setup and Formulation

Problem setup Consider a neural network with a loss function $\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N} \ell_i(w)$. Here, $w \in \mathbb{R}^p$ is the vector of trainable parameters, and N is the number of data points (samples). Given a pre-trained weight vector $\bar{w}$, our goal is to set some parameters to zero while maintaining the original model's performance (e.g., accuracy) as much as possible. In mathematical terms, given $\bar{w} \approx \arg\min_w \mathcal{L}(w)$ and a target level of sparsity $\tau \in (0,1)$, we hope to construct $w \in \mathbb{R}^p$ that satisfies (a) (loss function after pruning is close to that before pruning) $\mathcal{L}(w) \approx \mathcal{L}(\bar{w})$; and (b) (budget on number of nonzero weights after pruning) $\|w\|_0 \leq (1-\tau)p$. Here, we focus on one-shot pruning where no retraining or fine-tuning based on SGD is performed after pruning[1] to make it easier to compare our results with existing work, e.g. [30, 35].

Problem formulation. Following previous work [16, 22, 30, 35], the loss function can be locally approximated around $\bar{w}$ as (1). With this approximation, the network pruning problem can be formulated as the following MIQP

$$\min_w \; f(w; \bar{w}, g, H, \lambda) \quad \text{s.t.} \;\; \|w\|_0 \leq k, \tag{2}$$

where $k = \lfloor (1-\tau)p \rfloor$ and $f(w; \bar{w}, g, H, \lambda) := g^\top (w-\bar{w}) + 0.5(w-\bar{w})^\top H(w-\bar{w}) + (\lambda/2)\|w-\bar{w}\|^2$ with $\lambda \geq 0$ and certain choices of gradient and Hessian approximations $g \approx \nabla \mathcal{L}(\bar{w}), H \approx \nabla^2 \mathcal{L}(\bar{w})$. In the MIQP (2), we minimize the quadratic approximation to make sure the pruned model remains as close as possible to the dense model, while enforcing the sparsity constraint.

Ridge term and choices of λ. The performance of our final pruned model depends crucially on the validity of the quadratic approximation of the loss function. This approximation is a local one, and it is therefore important to ensure that the iterates (during the course of the algorithm) are close enough to $\bar{w}$. One way to ensure this is to include a squared ℓ_2 penalty on $w - \bar{w}$ (also known as the ridge or Tikhonov regularization). In previous works on pruning using the local quadratic approximations [16, 30, 35], such a penalty is not considered explicitly, but is used implicitly to obtain an invertible Fisher matrix, by adding a small diagonal λI to the Fisher matrix[2]. Additional investigation into the effect of λ is provided in Appendix B.2.1.

Choices of Hessian approximation H. Following previous works [16, 30], we approximate the Hessian matrix by the empirical Fisher information matrix, using n samples. This leads to a low-rank expression of $H = (1/n)\sum_{i=1}^n \nabla \ell_i(\bar{w}) \nabla \ell_i(\bar{w})^\top = (1/n) X^\top X \in \mathbb{R}^{p \times p}$, where $X = [\nabla \ell_1(\bar{w}), \ldots, \nabla \ell_n(\bar{w})]^\top \in \mathbb{R}^{n \times p}$. As shown later, H is never computed in our approach as we only require X. Previous works have noted that it is sufficient to use a few hundred to one thousand samples to estimate the Hessian—this aligns with what we observe in our studies.

Choices of gradient approximation g. Most of the previous works assume that the pre-trained weights $\bar{w}$ are a local optimum of $\mathcal{L}$, 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 the use of early stopping (or approximate optimization) [33]. Also, it might be more beneficial to prune the local quadratic model computed at a general reference point (where gradient is not zero) — this is an approach we follow in our multi-stage procedure. Therefore, we propose to approximate the gradient by the stochastic gradient, using the same samples for estimating the Hessian. That is $g = (1/n)\sum_{i=1}^n \nabla \ell_i(\bar{w}) = (1/n) X^\top e \in \mathbb{R}^p$, where $e \in \mathbb{R}^n$ is a vector of all ones. As we discuss in Appendix B.2.2, this first order term needs to be properly scaled, especially when taking losses as averages from batches of size more than one. In the WoodTaylor approach [30], this scaling issue is handled by tuning the dampening factor λ. We instead propose scaling the first order term as the inverse of the batch size (See Appendix B.2.2 for more details).

1. The procedure of iteratively pruning and then fine-tuning using SGD is called gradual pruning—see for eg [30].
2. The λ in previous work is usually taken in the range $[10^{-6}, 10^{-4}]$.

ℓ_0-regression formulation. With these approximations of g and H,we can now formulate the MIQP (2) into the following ℓ_0-constrained ridge regression problem:

$$\min_{w} \; Q(w) := \frac{1}{2}\|y - Xw\|^2 + \frac{n\lambda}{2}\|w - \bar{w}\|^2, \;\; \text{s.t.} \;\; \|w\|_0 \le k, \tag{3}$$

for $y = X\bar{w} - e \in \mathbb{R}^n$ and $X \in \mathbb{R}^{n\times p}$. Note that our algorithmic framework also applies to (3) with a general y and X, and in particular to the case without a first-order term (i.e. $g = 0$) by taking $y = X\bar{w}$ instead. Crucially, the regression formulation (3) does not require us to compute or store the full Hessian matrix $H \in \mathbb{R}^{p\times p}$, and as we will show in the next section, we only need to operate on the matrix X throughout our algorithm, which is the source of substantial gains in terms of both memory consumption and runtime.

3. Algorithmic Framework for Solving problem (3)

In this section, we present main ideas of our proposed algorithms for solving the ℓ_0-constrained regression problem (3), with additional details presented in Appendix A. Our primal goal is to develop efficient algorithms that converge rapidly, as in practice the scale of the network could be too big ($p \sim 10^6$) to be well optimized.

Single-stage methods We design first-order algorithms based on the framework of iterative hard thresholding (IHT, [4, 7]), together with several novel strategies and tricks to accelerate convergence:

- **IHT with aggressive stepsize**: The traditional choices of stepsize of IHT is either too conservative or even hard to compute. We propose a novel step size scheme which yield significantly more aggressive step size (about 10 times), based on the fact that the line search objective is a piecewise quadratic function and the optimal stepsize on each piece can be computed exactly. We detail the ideas and algorithms in Appendix A.1.
- **Refining on the support**: IHT algorithm does quite well in identifying a good support. After the support is identified, we propose to use following methods to refine the solution on the support to further decrease the objective: (i) Coordinate descent (CD, [3, 28]); (ii) Back solve based on Woodbury formula [25]. The details about our refining (or polishing) strategies are provided in Appendix A.2 and A.4.
- **Active set updates**: To further reduce the $O(np)$ iteration-complexity of IHT, we propose to use the active set strategy, which has turned out to be an effective tool in several other contexts [5, 8, 17]. The algorithm details can be found in Appendix A.3.

Multi-stage procedure The above single-stage methods are quite fast and memory-efficient, and they can achieve comparable results of model performance compared to the existing methods. In addition to these methods, we propose a multi-stage procedure where we update and solve local quadratic models sequentially. Furthermore, to ensure the validity of the local quadratic approximation, we use a dense-to-sparse scheduler on the sparsity constraint and take a small sparsity step in each stage (see Algorithm 1 for more details).

4. Experimental Results

In this section, we compare our methods with several previous pruning methods on different pretrained networks. The existing methods we consider are MP (magnitude pruning [27]), WF (Wood-

Algorithm 1 Multi-stage pruning

```
Require: Pre-trained weights $\bar{w}$, target number of nonzeros $k$, number of stages $m$.
1: Initialization: Construct a decreasing sparse mesh $k_1, k_2, \dots, k_f = k$; $w^0 = \bar{w}$
2: for $t = 1, 2, \dots, f$ do
3:     Update $y$ and $X$ in (3) based on current solution $w^{t-1}$
4:     Obtain a solution $w^t$ to Problem (3) with $\bar{w} = w^{t-1}$ and $k = k_t$ using single-stage approach
5: end for
```

Fisher [30]) and CBS (Combinatorial Brain Surgeon [35]). The pre-trained networks are MLP-Net (30k parameters) trained on MNIST [23], ResNet20 ([18], 200k parameters) trained on CIFAR10 [20], and MobileNetV1 (4.2M parameters) trained on ImageNet [9]. All these networks are considered in [35]. More details for reproducibility are provided in Appendix B.1.

Runtime comparison [30, 35] note that using more samples tend to have better accuracy, which is also something that we notice in our experiments. However, most prior approaches become computationally expensive as n becomes big. Here, we compare the runtime of our approach to that of M-FAC [12], a well optimized and more efficient implementation of Woodfisher [30]. As shown in Figures 1a and 1b, our algorithm offers significant speedups. Note that as is reported in [12], M-FAC is at least 100 times faster than WoodFisher; as for CBS, it takes a few hours on average to prune the MobileNetV1 network in contrast to less than one minute using our approach. This means our methods achieve up to 1,000 times faster runtime compared to WoodFisher and CBS.

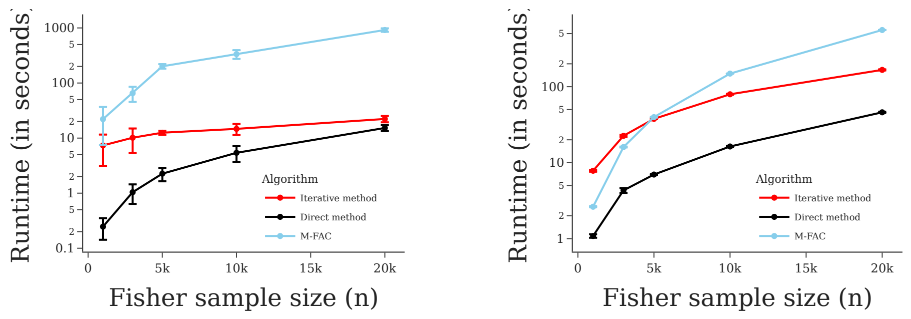
Comparison of model performance Table 1 compares the test accuracies of MLPNet, Resnet20 and MobileNetV1 pruned to different sparsity levels. Our single-stage method achieves comparable results to other state-of-the-art approaches with much less time consumption. The multi-stage method stated in Algorithm 1 outperforms other methods by a large margin, especially with a high sparsity rate.

Other results In addition to the above results, we conduct several ablation studies in Appendix B.2. In Appendix B.2.1, we explore the effects of the ridge term on the out-of-sample accuracy. We provide in Appendix B.2.2 a heuristic strategy to determine the scaling factor of the first-order term, along with several experiments to verify the effectiveness of the first order term together with this strategy. The importance of choosing appropriate sparsity schedules in the multi-stage approach is illustrated in Appendix B.2.3.

5. Conclusion and Ongoing/Future Work

In summary, we propose an efficient network pruning approach based on an ℓ_0-constrained regression problem and novel combinatorial optimization techniques. Compared to the existing methods, our single-stage methods achieve comparable results and run significantly faster with much less memory usage. Built upon the single-stage methods, our multi-stage method can make further large improvements in the model accuracy.

As future work, it would be interesting to test our framework on larger problems, such as ResNet50 (25M parameters, [18]) and BERT (110M parameters, [10]). Additionally, we are looking into the setting of gradual pruning like [30].



(a) CPU time on pruning MLPNet to 90% sparsity level

(b) CPU time on pruning Resnet20 to 90% sparsity level

Figure 1: Runtime comparison between our single-stage approach and M-FAC. The error bar represents the standard error over 4 runs. Iterative method and Direct method are two of our single-stage approaches. See Algorithm 4 and Algorithm 5 in Appendix A for detailed descriptions.

Network	Sparsity	MP	WF	CBS	Our method (single stage)	Our method (multi-stage)
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	90.6 (±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	57.9 (±1.04)	**88.72** (±0.17)
	0.9	11.79	11.49	13.68	15.6 (±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 and MobileNetV1. As to the performance of MP, WF and CBS, we adopt the results reported in [35]. We take 5 runs for our single-stage and multi-stage approaches, and report the mean and standard error (in the brackets). The accuracy of dense models (without pruning) are provided in the leftmost column. The best accuracy numbers (within 0.01 difference) are highlighted in bold.

6. Acknowledgement

This research is supported in part by the Office of Naval Research (N000142112841 and N000142212665). We thank Shibal Ibrahim for helpful discussions. We also thank Thiago Serra and Yu Xin for sharing with us code from their CBS paper [35].

References

[1] Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. *Advances in neural information processing systems*, 30, 2017.

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

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

[4] Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *The annals of statistics*, 44(2):813–852, 2016.

[5] Kush Bhatia, Prateek Jain, and Purushottam Kar. Robust regression via hard thresholding. *Advances in neural information processing systems*, 28, 2015.

[6] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

[7] Thomas Blumensath and Mike E Davies. Iterative hard thresholding for compressed sensing. *Applied and computational harmonic analysis*, 27(3):265–274, 2009.

[8] Wenyu Chen and Rahul Mazumder. Multivariate convex regression at scale. *arXiv preprint arXiv:2005.11588*, 2020.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[11] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in Neural Information Processing Systems*, 30, 2017.

[12] Elias Frantar, Eldar Kurtic, and Dan Alistarh. M-fac: Efficient matrix-free approximations of second-order information. *Advances in Neural Information Processing Systems*, 34:14873–14886, 2021.

[13] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[14] Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*, 2020.

[15] Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems*, 1, 1988.

[16] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

[17] Hussein Hazimeh and Rahul Mazumder. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. *Operations Research*, 68(5):1517–1537, 2020.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] Rafael Lazimy. Mixed-integer quadratic programming. *Mathematical Programming*, 22(1): 332–349, 1982.

[22] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[25] A Woodbury Max. Inverting modified matrices. In *Memorandum Rept. 42, Statistical Research Group*, page 4. Princeton Univ., 1950.

[26] Rahul Mazumder, Peter Radchenko, and Antoine Dedieu. Subset selection with shrinkage: Sparse linear modeling when the snr is low. *Operations Research*, 2022.

[27] Michael C Mozer and Paul Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16, 1989.

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

[29] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117, 2015.

[30] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109, 2020.

[31] Valentin Thomas, Fabian Pedregosa, Bart Merriënboer, Pierre-Antoine Manzagol, Yoshua Bengio, and Nicolas Le Roux. On the interplay between noise and curvature and its effect on optimization and generalization. In *International Conference on Artificial Intelligence and Statistics*, pages 3503–3513. PMLR, 2020.

[32] Dongxian Wu and Yisen Wang. Adversarial neuron pruning purifies backdoored deep models. *Advances in Neural Information Processing Systems*, 34:16913–16925, 2021.

[33] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

[34] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael Mahoney. Pyhessian: Neural networks through the lens of the hessian. In *ICML workshop on Beyond First-Order Optimization Methods in Machine Learning*, 2020.

[35] Xin Yu, Thiago Serra, Srikumar Ramalingam, and Shandian Zhe. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In *International Conference on Machine Learning*, pages 25668–25683. PMLR, 2022.

Appendix A. Algorithmic details

A.1. IHT with aggressive stepsize

IHT algorithm Our optimization framework relies on the IHT algorithm to identify the support as well as make descent steps. Each step of the IHT algorithm takes a gradient step with step size τ from the current iterate and then projects it onto the cardinality constraint set by 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, and the hard thresholding operator P_k is defined such that $P_k(x) = y$ has

$$y_i = \begin{cases} x_i, & \text{if } i \in \mathcal{I}_k(x), \\ 0, & \text{otherwise} \end{cases}$$

IHT applied to problem (3) renders the following updating formula

$$w^{t+1} = \mathsf{HT}(w^t, k, \tau) := P_k\left(w^t - \tau\nabla Q(w^t)\right) = P_k\left(w^t - \tau(X^\top(Xy - w^t) + n\lambda(w^t - \bar{w}))\right), \tag{4}$$

where $\tau > 0$ is a stepsize to be determined. The computation of $\mathsf{HT}(w^t, k, \tau)$ involves only the matrix-vector multiplications between X (or its transpose) and a vector, which has a total computation cost of $O(np)$. This is a significant reduction compared to the $O(p^2)$ cost of using the full Hessian matrix as [30, 35] do.

Challenges of stepsize choice The choice of the step size τ is a big challenge here. In theory, setting τ as the constant $1/L$ in (4) is a common choice in the literature to ensure the convergence to a stationary solution [4, 17], where L is the Lipschitz constant of the gradient of $Q(w)$. i.e., $\|\nabla Q(\alpha) - \nabla Q(\beta)\| \le L\|\alpha - \beta\|$ for all $\alpha,\ \beta \in \mathbb{R}^p$. Since Q is a quadratic objective, this quantity L is given by $L = n\lambda + \|X\|_{op}^2$, where $\|X\|_{op}$ is the operator norm of X, i.e., the leading singular value of X. In practice, when p is large, this quantity could be very large, leading to very conservative updates, sometimes negligible. Moreover, the computation of L itself may involve a few power iterations or randomized SVD algorithm, which could be as costly as several iterations of IHT updates. Another approach is through line search, e.g. [2]. The basic idea is to try different values of τ to find the best τ that makes the objective descent as steep as possible. This requires multiple evaluations of the quadratic. We observe that updates given by vanilla line search methods are still conservative and consequently the support of the iterates does not change most of time.

Our novel scheme To this end, we present a novel scheme to determine a much more aggressive stepsize. We notice, from the definition of P_k, that $g(\tau) := Q\left(P_k\left(w^t - \tau\nabla Q(w^t)\right)\right)$ is a univariate piecewise quadratic function, and the goal of line search is to find a value τ that makes g as small as possible. Furthermore, denote by τ_c the largest value of τ' such that the hard thresholding based on $\tau \in [0, \tau']$ does not change the support, i.e. $\mathcal{I}_k(w^t) = \mathcal{I}_k(w^t - \tau\nabla Q(w^t)),\ \forall\tau \in [0, \tau']$. Then we know that over $\tau \in [0, \tau_c]$, $g(\tau)$ is a quadratic function, and let τ_m denote the minimizer of $g(\tau)$ over $[0, \tau_c]$. If $\tau_m < \tau_c$, then we use $\tau = \tau_m$ as stepsize; otherwise, we performe a line search by setting $\tau \leftarrow \gamma\tau$ $(\gamma > 1)$ starting from τ_c to determine the stepsize with steepest descent. Algorithm 2 summarizes the above procedure. Intuitively, this helps avoid the redundant line search steps on the quadratic piece of $g(\tau)$ over $[0, \tau_c]$, which results in a more aggressive step size.

In terms of overall complexity, it is not hard to see that τ_c can be computed in $O(p)$ after we obtain the gradient $\nabla Q(w^t)$; and the cost of computing τ_m is the same as the quadratic objective evaluation, which is already required by the line search.

Finally, we note that during the course of line search, it is always possible to find the piece of the quadratic function to which the current stepsize τ belongs, say $[\tau_l, \tau_u]$, 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 usually the line search procedure terminates in few steps.

Algorithm 2 A novel scheme to determine the stepsize τ

```
Require: $w^t$, $k$, $\gamma > 1$
 1: Set $\tau_c$ as the largest value such that $\mathcal{I}_k(w^t) = \mathcal{I}_k(w^t - \tau'\nabla Q(w^t))$, $\forall \tau' \in [0, \tau_c]$.
 2: Compute
        $$\tau_m = \underset{\tau' \in [0,\tau_c]}{\arg\min}\, g(\tau') := Q\left(P_k\left(w^t - \tau'\nabla Q(w^t)\right)\right) \tag{5}$$
 3: if $\tau_m < \tau_c$ then
 4:     $\tau \leftarrow \tau_m$.
 5: else
 6:     $g_{\text{best}} \leftarrow g(\tau_c)$, and $\tau \leftarrow \tau_c$.
 7:     while $g_{\text{best}} > g(\gamma\tau)$ do
 8:         $g_{\text{best}} \leftarrow g(\gamma\tau)$, and $\tau \leftarrow \gamma\tau$.
 9:     end while
10: end if
```

A.2. Cyclic coordinate descent

Although IHT does well in identifying and updating the support, in experiments we see that it makes slow progress in decreasing the objective. To this end, in order to decrease the objective faster, we use cyclic coordinate descent (CD, [3, 28]) 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 [13] and L_0L_2-regularized regression [17].

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:

$$w_i^{t+1} = \mathsf{CDUpdate}(w^t, i) := \underset{w \in \mathbb{R}}{\arg\min}\, Q\left(w_1^t, \ldots, w_{i-1}^t, w, w_{i+1}^t, \ldots, w_p^t\right). \tag{6}$$

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

Cyclic CD enjoys a fast convergence rate [3, 28]. However, the quality of the resulting solution is limited and depends highly on the initial solution, as CD cannot modify the support of a solution. For a better performance in terms of both quality and efficiency, in practice we adopt a hybrid updating rule that combines IHT and cyclic CD. 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: $\mathsf{IHT\text{-}CD}(w^0, k, t_{\mathsf{HT}}, t_{\mathsf{CD}})$

```
Require: $w^0$, $k$, $t_{\mathsf{HT}}, t_{\mathsf{CD}}$
 1: for $t = 0, 1, \dots$ do
 2:     $w \leftarrow w^t$
 3:     for $t = 1, \dots, t_{\mathsf{HT}}$ do
 4:         $w \leftarrow \mathsf{HT}(w, k, \tau)$                    ▷ Time complexity $O(np)$
 5:     end for
 6:     for $\tau = 1, \dots, t_{\mathsf{CD}}$ do
 7:         for $i \in \mathrm{supp}(w)$ do
 8:             $w \leftarrow \mathsf{CDUpdate}(w, i)$               ▷ Time complexity $O(n)$
 9:         end for
10:     end for
11:     $w^{t+1} \leftarrow w$
12: end for
```

A.3. Active set updates

Previous works [5, 8, 17] have shown success of using the active set strategies in various contexts. We use such approaches as well to make our algorithm more efficient. In our problem setting, the active set strategy works as follows: an initial active set (of length equal to a multiple of the required sparsity k, e.g. $2k$) is selected based on the magnitude of the initial solution. In each iteration, we restricts the update of Algorithm 3 to the current active set $\mathcal{A}$. After convergence, we perform IHT updates on the full vector to search for 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 solver, e.g. Algorithm 5 in later section). In our experiments, such strategy works really well on small or middle-sized problems ($p \sim 10^5$) and sparse problems ($k \ll p$).

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

```
Require: $w^0$, $k$, $t_{\mathsf{HT}}, t_{\mathsf{CD}}$, and an initial active set $\mathcal{A}^0$
 1: for $m = 0, 1, \dots$ do
 2:     $w^{m+1/2}|_{\mathcal{A}^m} \leftarrow \mathsf{IHT\text{-}CD}(w^m|_{\mathcal{A}^m}, k, t_{\mathsf{HT}}, t_{\mathsf{CD}})$ restricted on $\mathcal{A}^m$
 3:     Find $\tau$ via line search such that $w^{m+1} \leftarrow \mathsf{HT}(w^{m+1/2}, k, \tau)$ satisfies
 4:       (i) $Q(w^{m+1}) < Q(w^{m+1/2})$  (ii) $\mathrm{supp}(w^{m+1}) \not\subseteq \mathcal{A}^m$
 5:     if such $\tau$ does not exist then
 6:         break
 7:     else
 8:         $\mathcal{A}^{m+1} \leftarrow \mathcal{A}^m \cup \mathrm{supp}(w^{m+1})$
 9:     end if
10: end for
```

A.4. Backsolve via Woodbury formula

As p and k become larger, the active set method becomes more costly. Hence, we propose a backsolve approach that further reduces the complexity, at the cost of a slightly worse solution. The backsolve approach calculates the optimal solution exactly on a restricted set. We first apply IHT updates a few number of times to get an initial feasible solution w, and then restrict the problem (3) to the set $\mathcal{S} := \operatorname{supp}(w)$. Under the restriction, problem (3) reduces to a quadratic problem without ℓ_0 constraint and it's minimizer reads

$$w^*_{\mathcal{S}} = (n\lambda I_k + X_{\mathcal{S}}^\top X_{\mathcal{S}})^{-1}(n\lambda \bar{w}_{\mathcal{S}} + X_{\mathcal{S}}^\top y_{\mathcal{S}}), \tag{7}$$

where $X_{\mathcal{S}} \in \mathbb{R}^{n\times k}$ is a submatrix of X with columns only in $\mathcal{S}$. By exploiting the low-rank structure of $X_{\mathcal{S}}^\top X_{\mathcal{S}}$ and utilizing Woodbury formula [25], (7) can be computed in $O(n^2k)$ operations. Specifically, one can compute (7) using matrix-vector multiplications involving only $X_{\mathcal{S}}$ (or its transpose) and one matrix-matrix multiplication via

$$\begin{aligned} w^*_{\mathcal{S}} &= (n\lambda)^{-1}[I_k - X_{\mathcal{S}}^\top(n\lambda I_k + X_{\mathcal{S}}X_{\mathcal{S}}^\top)^{-1}X_{\mathcal{S}}]\cdot(n\lambda\bar{w}_{\mathcal{S}} + X_{\mathcal{S}}^\top y_{\mathcal{S}}) \\ &= \bar{w}_{\mathcal{S}} + (n\lambda)^{-1}X_{\mathcal{S}}^\top y_{\mathcal{S}} - (n\lambda)^{-1}X_{\mathcal{S}}^\top(n\lambda I_k + X_{\mathcal{S}}X_{\mathcal{S}}^\top)^{-1}X_{\mathcal{S}}(n\lambda\bar{w}_{\mathcal{S}} + X_{\mathcal{S}}^\top y_{\mathcal{S}}). \end{aligned} \tag{8}$$

The backsolve method is stated in Algorithm 5.

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

```
Require: $w^0$, $k$, $t_{\mathsf{HT}}$.
1: for $t = 1, \ldots, t_{\mathsf{HT}}$ do
2:     $w \leftarrow \mathsf{HT}(w^0, k, \tau)$                                  ▷ Time complexity $O(np)$
3: end for
4: $\mathcal{S} \leftarrow \operatorname{supp}(w)$
5: $w_{\mathcal{S}} \leftarrow (n\lambda I_k + X_{\mathcal{S}}^\top X_{\mathcal{S}})^{-1}(n\lambda\bar{w}_{\mathcal{S}} + X_{\mathcal{S}}^\top y_{\mathcal{S}})$, using (8)   ▷ Time complexity $O(n^2k)$
```

We note that prior works [16, 30, 35] also use the formula, but 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 (3) can then be decomposed into the following subproblems ($1 \le i \le c$)

$$\min_{w\in\mathbb{R}^{|B_i|}} \frac{1}{2}\|y_i - X_{B_i}w\|^2 + \frac{n\lambda}{2}\|w - \bar{w}_{B_i}\|^2, \ \text{ s.t. } \|w\|_0 \le k_i, \tag{9}$$

where $y_i = X_{B_i}\bar{w}_{B_i} - e$ 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 4 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 [30, 35]. 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}.

We clarify that the introduction of block-wise approximation is for the sake of solution quality rather than efficiency. This differs to previous works [30, 35]. In fact, solving (9) for $i = 1, \dots, c$ requires operations of the same order as solving (3) directly. On the other side, we observe in our experiments that adopting block-wise approximation will increase the accuracy of the pruned MobileNetV1 network dramatically (from 0.2% to near 30% at sparsity level 0.8).

Algorithm 6 Back solve with block approximation

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

Appendix B. Experiment details

B.1. Experimental setup

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 were run on an Intel Xeon Gold 6248 machine with 40 CPUs and one GPU.

MLPNet Architecture The considerd MLPNet has three linear layers of architecture $3072 \to 40 \to 20 \to 10$, following [35].

Algorithmic setting We adopt the active set strategy with IHT and CD update (Algorithm 4) to prune MLPNet and ResNet20. We also set Algorithm 4 as the inner solver in Algorithm 1 for these two networks. For MobileNet, we utilize the direct back solve method with block approximation (Algorithm 6) as the single-stage approach. We employ the same block-wise approximation as in the prior work [30, 35], see Section A.5 for details. We choose Algorithm 5 as the inner solver when performing multi-stage approach (Algorithm 1) on the MobileNet. For all multi-stage approaches, we use 15 stages.

Fisher sample size and mini-batch size We report in Table 2 the Fisher sample size n and the mini-batch size m used for gradient evaluation in Hessian approximation (i.e., in Hessian approximation, each $\nabla \ell_i(\bar{w})$ is replaced by the average gradient of a mini-batch of size m). Note that for MLPNet and ResNet20, WF, CBS as well as our method make use of the same amount of gradients; while for MobileNet, our method performs $16{,}000$ gradient evaluations, which is much less compared to WF and CBS as they require $960{,}000$ gradient evaluations.

Hyper-parameters For each network and each sparsity level, we run our single-stage and multi-stage methods with ridge value λ ranging from $[10^{-5}, 10^3]$ and number of IHT iterations (if Algorithm 4 is applied) ranging from $[5, 500]$. We also try single-stage algorithm with/without the first

Model	MLPNet		Resnet20		MobileNet	
	sample	batch	sample	batch	sample	batch
WF [30] & CBS[35]	1000	1	1000	1	400	2400
Our method	1000	1	1000	1	1000	16

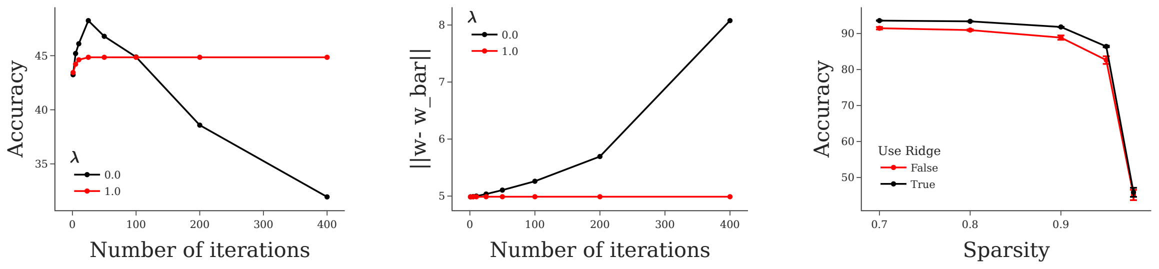
Table 2: Comparisons of the Fisher sample size n and the mini-batch size m used for gradient evaluation in Hessian approximation on MLPNet, ResNet20 and MobileNet.

order term. We report in Table 1 the best model accuracy over all possible hyper-parameter combinations. For Resnet20, we use the same block approximation used by CBS, by taking a block for each layer. As for MobileNetV1, we use a block size of 2000, as this yields the best results.

B.2. Implementation details and ablation study

B.2.1. Ridge term

We study the effect of the ridge term and display some experimental results in Figure 2. We observe from Figure 2b that without a ridge term, the distance between $\bar{w}$ and the pruned weight w becomes larger as algorithm processes. As larger distance leads to poorer local quadratic approximation, the performance of model without the ridge term starts to decrease after dozens of iterations, as shown in Figure 2a. Figure 2c demonstrates some additional evidence on the effectiveness of ridge term in improving the accuracy. We conclude from these figures that including a ridge term is of great importance in getting stable models with high performance.



(a) Accuracy w.r.t. the number of iterations in a run of Algorithm 4

(b) $||w - \bar{w}||$ w.r.t. the number of iterations in a run of Algorithm 4

(c) Effect of using ridge term (error bars are standard errors over 4 runs).

Figure 2: Effect of the ridge term. We apply Algorithm 4 to prune MLPNet and report the results. In Figure 2a and 2b, the network is pruned to a 98% sparsity level.

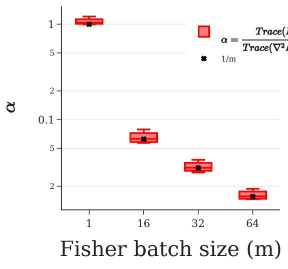
B.2.2. First-order term and the scaling factor

It has been shown in recent works [30, 31] that the empirical Fisher matrix $H = (1/n)X^\top X$ has a high cosine similarity with the true Hessian, and is therefore a good approximation up to a scaling factor, i.e., there exists $\alpha > 0$ such that $H \approx \alpha\nabla^2\mathcal{L}(\bar{w})$. For scale-independent applications, e.g., MIQP considered in (2) without the first order term, the empirical Fisher matrix H could be regarded as a good and cheap estimation of the Hessian. On the other side, Figure 3b indicates that we have to estimate the scaling factor α when the first order term is involved (in this case the problem is no longer scale-independent), otherwise the model yields poor results..

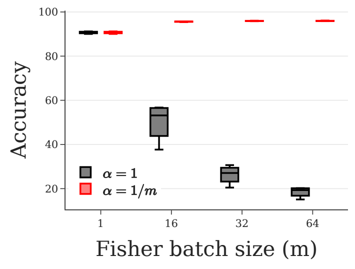
Using the code provided by [34], we can compute an estimate of the scaling factor α using

$$\alpha = \frac{Trace(H)}{Trace(\nabla^2 \mathcal{L}(\bar{w}))}. \tag{10}$$

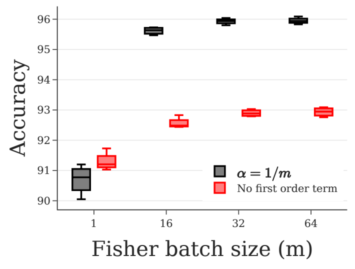
However, the computation cost of $Trace(\nabla^2 \mathcal{L}(\bar{w}))$ is not negligible. As shown in Figure 3a, we observe in our experiments that α as estimated by (10) is close to $1/m$, where m is the mini-batch size used for gradient evaluation in Hessian approximation. Hence, we set $\alpha = 1/m$ as a heuristic scaling factor in our experiments. Figure 3b and 3c demonstrate the advantage of using scaled first order term over incorrect scaling and not using first order term.



(a) Difference between α calculated by (10) and our heuristic



(b) Accuracy w.r.t. size of a mini batch



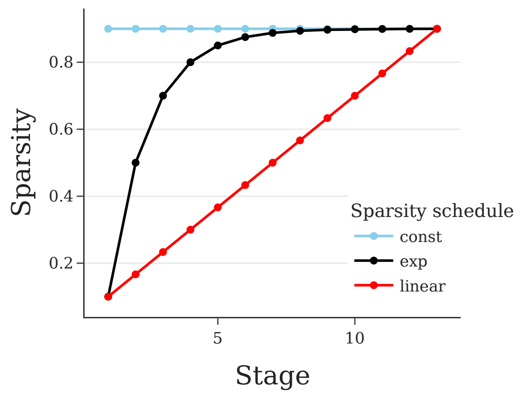
(c) Accuracy w.r.t. size of a mini batch

Figure 3: Effect of using scaled first order term. All the results are averaged over 5 runs of pruning MLPNet to 95% sparsity rate. We utilize our multi-stage approach with Algorithm 4 as the inner solver.

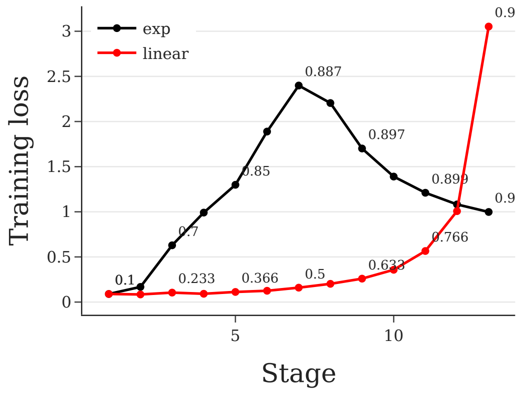
B.2.3. SPARSITY SCHEDULE IN MULTI-STAGE PROCEDURE

We observe in our experiments that the performance of the multi-stage algorithm depends highly on the choice of the sparsity mesh $k_1 \geq k_2 \geq \cdots \geq k_f = k$. We compare in this section test accuracies between two different schedules: (i) an exponential mesh ($k_t - k_{t-1} \sim \frac{1}{2^t}$) and (ii) a linear mesh ($k_t - k_{t-1} \sim 1$). As shown in Figure 4a, the exponential mesh takes smaller steps as the sparsity level increases, whereas the linear schedule takes constant steps.

Figure 4b plots the training loss of the two schedules in each stage. We see from the figure that the performance of the linear mesh drops dramatically in the last few iterations, around the sparsity level 0.766. Taking small stepsizes in high sparsity levels allows the exponential mesh to maintain a good accuracy in the challenging sparsity levels, achieving good performance.



(a) Two different sparsity schedules: exponential mesh and linear mesh.



(b) Training loss w.r.t. number of stages during the pruning process. Text around the point indicates the current sparsity level of the point.

Figure 4: Effect of different sparsity schedules on model accuracy. We utilize our multi-stage approach with Algorithm 4 as the inner solver for pruning MLPNet to 90% sparsity rate.