LOOKAHEAD: A FAR-SIGHTED ALTERNATIVE OF MAGNITUDE-BASED PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Magnitude-based pruning is one of the simplest methods for pruning neural networks. Despite its simplicity, magnitude-based pruning and its variants have shown state-of-the-art performances for pruning modern architectures. Based on the observation that the magnitude-based pruning indeed minimizes the Frobenius distortion of a linear operator corresponding to a single layer, we develop a simple pruning method, coined lookahead pruning, by extending the single layer optimization to a multi-layer optimization. Our experimental results demonstrate that the proposed method consistently outperforms the magnitude pruning on various networks including VGG and ResNet, particularly in the high-sparsity regime.

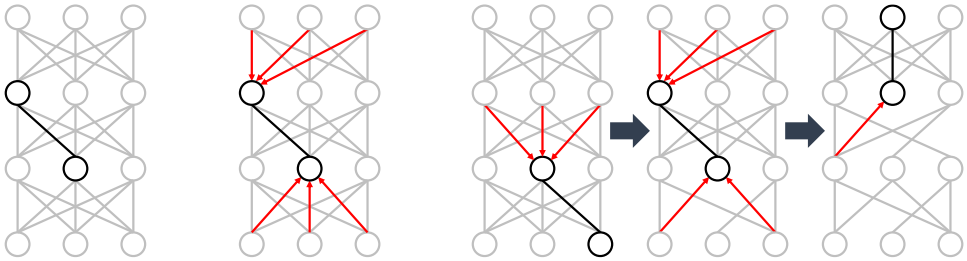
1 INTRODUCTION

The "magnitude-equals-saliency" approach has been long underlooked as an overly simplistic baseline among all imaginable techniques to eliminate unnecessary weights from over-parametrized neural networks. Since the early works of LeCun et al. (1989); Hassibi & Stork (1993) which provided more theoretically grounded alternative of magnitude-based pruning (MP) based on second derivatives of loss function, a wide range of methods including Bayesian / information-theoretic approaches (Neal, 1996; Louizos et al., 2017; Molchanov et al., 2017; Dai et al., 2018), ℓ_p-regularization (Wen et al., 2016; Liu et al., 2017; Louizos et al., 2018), sharing redundant channels (Zhang et al., 2018; Ding et al., 2019), and reinforcement learning approaches (Lin et al., 2017; Bellec et al., 2018; He et al., 2018) has been proposed as more sophisticated alternatives.

On the other hand, the capabilities of MP heuristics are gaining attention once more. Combined with minimalistic techniques including iterative pruning (Han et al., 2015) and dynamic reestablishment of connections (Zhu & Gupta, 2017), a recent large-scale study by Gale et al. (2019) has demonstrated that MP can achieve state-of-the-art trade-off of sparsity and accuracy. The unreasonable effectiveness of magnitude scores often extend beyond the strict domain of network pruning; a recent experiment by Frankle & Carbin (2019) suggests existence of an automatic subnetwork discovery mechanism underlying the standard gradient-based optimization procedures of deep, over-parametrized neural networks by showing that the MP algorithm finds an efficient trainable subnetwork. These observations constitute a call to revisit the "magnitude-equals-saliency" approach for a better understanding of deep neural network itself.

As an attempt to better understand the nature of MP methods, we study a generalization of magnitude scores under a *functional approximation* framework; by viewing MP as a relaxed minimization of distortion in layerwise operators introduced by zeroing out parameters, we can consider a multi-layer extension of the distortion minimization problem. Minimization of the newly suggested distortion measure which 'looks ahead' the impact of pruning on neighboring layers gives birth to a novel pruning strategy, coined *lookahead pruning* (LAP).

In this paper, we focus on comparison of the proposed LAP scheme to its MP counterpart. We empirically demonstrate that LAP consistently outperforms the MP under various setups including linear networks, fully-connected networks, and deep convolutional networks. In particular, the LAP consistently enables more than $\times 2$ gain in the compression rate of the considered models, with increasing benefits under the high-sparsity regime. Apart from its performance, the lookahead pruning method enjoys additional attractive properties:



(a) MP (b) LAP (c) LAP-forward

Figure 1: An illustration of magnitude-based pruning (MP), lookahead pruning (LAP) and ordered version of LAP (LAP-forward). MP only consider a single weight while LAP also considers the effects of neighboring edges. LAP-forward iteratively applies LAP from the bottom to the top.

- *Easy-to-use:* The proposed LAP method is a simple model-agnostic, score-based approach and can be implemented via elementary tensor operations, introducing only a minimal computational overhead, comparable to the magnitude-based pruning methods. Unlike most Hessian-based methods, the proposed method does not rely on the availability of additional training batches except during the retraining phase. It also has no hyper-parameter to tune, in contrast to sophisticated training-based and optimization-based schemes.
- *Versatility:* As our method simply replaces the "magnitude-as-saliency" criterion with a lookahead alternative, it can be deployed jointly with algorithmic tweaks developed for magnitude-based pruning, such as iterative pruning and retraining (Han et al., 2015) or joint pruning and training with dynamic reconnections (Zhu & Gupta, 2017; Gale et al., 2019).

The remainder of this manuscript is structured as follows: In Section 2, we introduce a functional approximation perspective toward MP and motivate LAP and its variants as a generalization of MP for multiple layer setups; in Section 3 we explore the capabilities of LAP and its variants with simple models, then move on to apply LAP to larger-scale models.

2 LOOKAHEAD: A FAR-SIGHTED LAYER APPROXIMATION

We begin by a more formal description of the magnitude-based pruning (MP) algorithm (LeCun et al., 1989; Han et al., 2015). Given an L-layer neural network associated with weight tensors $W_1, \ldots, W_L$, the MP algorithm removes connections with smallest absolute weights from each weight tensor, until the desired level of sparsity has been achieved. This layerwise procedure is equivalent to finding a mask M whose entries are either 0 or 1, incurring a smallest *Frobenius distortion*, measured by

$$\min_{M:\|M\|_0=s} \|W - M \odot W\|_F \,, \tag{1}$$

where $\odot$ denotes the Hadamard product, $\|\cdot\|_0$ denotes the entry-wise ℓ_0-norm, and s is a sparsity constraint imposed by some operational criteria.

Aiming to minimize the Frobenius distortion (Eq. (1)), the MP algorithm naturally admits a functional approximation interpretation. For the case of a fully-connected layer, the maximal difference between the output from a pruned and an unpruned layer can be bounded as

$$\|Wx - M \odot Wx\|_2 \leq \|W - M \odot W\|_2 \cdot \|x\|_2 \leq \|W - M \odot W\|_F \cdot \|x\|_2. \tag{2}$$

A similar idea holds for convolutional layers. For the case of a two-dimensional convolution with a single input and a single output channel, the corresponding linear operator takes a form of a doubly block circulant matrix constructed from the associated kernel tensor (see, e.g., Goodfellow et al. (2016)). Here, the Frobenius distortion of doubly block circulant matrices can be controlled by the Frobenius distortion of the weight tensor of the convolutional layer.[1]

[1]The case of multiple input/output channel or non-circular convolution can be dealt with similarly using channel-wise circulant matrices as a block. We refer the interested readers to Sedghi et al. (2019).

Algorithm 1 Lookahead Pruning (LAP)

Input: Weight tensors $W_1, \ldots, W_L$ of a trained network, desired sparsities $s_1, \ldots, s_L$
Output: 0-1 masks $M_1, \ldots, M_L$
for $i = 1, \ldots, L$ **do**
    Compute $\mathcal{L}_i(w)$ according to Eq. (4) for all entry w of W_i
    Set $\tilde{w}_{s_i}$ as a s_i smallest element of $\{\mathcal{L}_i(w) : w \text{ is an entry of } W_i\}$
    Set $M_i = \mathbb{1}\{W_i - \tilde{w}_{s_i} \geq 0\}$
end for

2.1 Lookahead distortion as a block approximation error

The myopic optimization (Eq. (1)) based on the per-layer Frobenius distortion falls short even in the simplest case of the two-layer linear neural network with one-dimensional output, where we consider predictors of a form $\widehat{Y} = u^\top W x$ and try to minimize the Frobenius distortion of $u^\top W$ (equivalent to ℓ_2 distortion in this case). Here, if u_i is extremely large, pruning any nonzero element in the i-th row of W may incur a significant Frobenius distortion.

Motivated by this observation, we consider a *block approximation* analogue of the magnitude-based pruning objective Eq. (1). Given an L-layer linear network associated with weight tensors $W_1, \ldots, W_L$, let $\mathcal{J}(W_i)$ denote the Jacobian matrix corresponding to linear operator characterized by W_i. For pruning the i-th layer, we take into account the weight tensors of neighboring layers W_{i-1}, W_{i+1} in addition to the original weight tensor W_i. In particular, we propose to minimize the Frobenius distortion of the operator block $\mathcal{J}(W_{i+1})\mathcal{J}(W_i)\mathcal{J}(W_{i-1})$, i.e.,

$$\min_{M_i : \|M_i\|_0 = s_i} \left\| \mathcal{J}(W_{i+1})\mathcal{J}(W_i)\mathcal{J}(W_{i-1}) - \mathcal{J}(W_{i+1})\mathcal{J}(M_i \odot W_i)\mathcal{J}(W_{i-1}) \right\|_F . \tag{3}$$

An explicit minimization of the block distortion (Eq. (3)), however, is computationally intractable (see Appendix A for a more detailed discussion).

To avoid an excessive computational overhead, we propose to use the following score-based pruning algorithm, coined lookahead pruning (LAP), for approximating Eq. (3): For each entry w of W_i, we prune away the weights with the smallest value of lookahead distortion, defined as

$$\mathcal{L}_i(w) := \left\| \mathcal{J}(W_{i+1})\mathcal{J}(W_i)\mathcal{J}(W_{i-1}) - \mathcal{J}(W_{i+1})\mathcal{J}(W_i|_{w=0})\mathcal{J}(W_{i-1}) \right\|_F \tag{4}$$

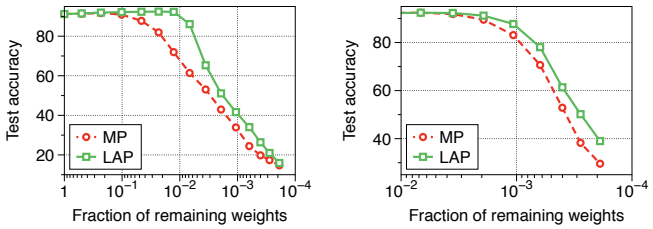
where $W_i|_{w=0}$ denotes the tensor whose entries are equal to the entries of W_i except for having zeroed out w. We let both W_0 and W_{L+1} to be tensors consisting of ones. In other words, lookahead distortion (Eq. (4)) measures the distortion (in Frobenius norm) induced by pruning w while all other weights remain intact. For three-layer blocks consisting only of fully-connected layers and convolutional layers, Eq. (4) reduces to the following compact formula: for an edge w connected to the j-th input neuron and the k-th output neuron of the i-th layer,

$$\mathcal{L}_i(w) = |w| \cdot \left\| W_{i-1}[j, :] \right\|_F \cdot \left\| W_{i+1}[:, k] \right\|_F , \tag{5}$$

where $|w|$ denotes the weight of the neuron, $W[j, :]$ denotes the slice of W composed of weights connected to the j-th output neuron, and $W[:, k]$ denotes the same for the k-th input neuron. A formal description of LAP is presented in Algorithm 1.

LAP on linear networks. To illustrate the benefit of lookahead, we evaluate the performance of MP and LAP on a linear fully-connected network with a single hidden layer of 1,000 nodes, trained with MNIST image classification dataset. Fig. 2a and Fig. 2b depict the test accuracy of models pruned with each methods, before and after retraining steps.

As can be expected from the discrepancy between the minimization objectives (Eqs. (1) and (3)), networks pruned with LAP outperform networks pruned with MP at every sparsity level, in terms of its performance before a retraining phase. Remarkably, we observe that test accuracy of models pruned with LAP monotonically increases from 91.2% to 92.3% as the sparsity level increases, until the fraction of surviving weights reaches 1.28%. At the same sparsity level, models pruned with MP achieves only 71.9% test accuracy. We also observe that LAP leads MP at every sparsity level even after a retraining phase, with an increasing margin as we consider a higher level of sparsity.



(a) before retraining (b) after retraining

Figure 2: Test accuracy of pruned linear network under varying levels of sparsity, (a) before and (b) after a retraining phase. MP denotes magnitude-based pruning and LAP denotes the lookahead pruning. All reported points are averaged over 5 trials.

2.2 Lookahead distortion with nonlinear activation functions

Most neural network models in practice deploy nonlinear activation functions, e.g., rectified linear units (ReLU), between the fully-connected or convolutional layers. Although the lookahead distortion was originally defined without any consideration of nonlinear activation functions, the quantity $\mathcal{L}_i(w)$ remains relevant to the original block approximation point of view even for the networks with commonly used activation functions, especially when the network is severely over-parametrized. To see this, consider a case where one aims to prune a connection in the first layer of a two-layer fully-connected network with ReLU, i.e.,

$$x \mapsto W_2\sigma(W_1 x), \tag{6}$$

where $\sigma(x) = \max\{0, x\}$ is applied entrywise. Under the over-parametrized scenario, zeroing out a single weight may alter the activation pattern of connected neurons with only negligible probability, which allows one to decouple the probability of activation of each neuron from the act of pruning each connection. This enables us to approximate the root mean square distortion of the network output introduced by pruning w of W_1 by $\sqrt{p_j}\mathcal{L}_1(w)$, where j is the index of the output neuron that w is connected to, and p_j denotes the probability of activation for the j-th neuron.

Another theoretical justification of using the lookahead distortion (Eq. (5)) for neural networks with nonlinear activation functions comes from recent discoveries regarding the implicit bias imposed by training procedures using stochastic gradient descent. More specifically, Du et al. (2018) proves the following result, generalizing the findings of Arora et al. (2018): For any two neighboring layers of fully-connected neural network using positive homogeneous activation functions, the quantity

$$\|W_{i+1}[:, j]\|_2^2 - \|W_i[j, :]\|_2^2 \tag{7}$$

remains constant for any hidden neuron j over training via gradient flow. In other words, the total outward flow of weights is tied to the inward flow of weights for each neuron. This observation hints at the possibility of a relative undergrowth of weight magnitude of an 'important' connection, in the case where the connection shares the same input/output neuron with other 'important' connections. From this viewpoint, the multiplicative factors in Eq. (5) take into account the abstract notion of neuronal importance score, assigning significance to connections to the neuron through which more gradient signals have flowed through. Without considering such factors, LAP reduces to the ordinary magnitude-based pruning.

We perform empirical validations on the deployment of LAP on neural networks with nonlinear activation functions in Section 3.1, where we find that LAP is an effective strategy not only with piece-wise linear activation function like ReLU, but also with sigmoid or tanh functions.

2.3 Variants of lookahead pruning

As the LAP algorithm (Algorithm 1) takes into account current states of the neighboring layers, LAP admits several variants in terms of lookahead direction, order of pruning, and sequential pruning methods; these methods are extensively studied in Section 3.2 Along with "vanilla" LAP, we consider in total, six variants, which we now describe below:

Mono-directional LAPs. To prune a layer, LAP considers both preceding and succeeding layers. Looking forward, i.e., only considering the succeeding layer, can be viewed as an educated modification of the internal representation the present layer produces. Looking backward, on the other hand, can be interpreted as only taking into account the expected structure of input coming into the present layer. The corresponding variants, coined LFP and LBP, are tested.

Order of pruning. Instead of using the unpruned tensors of preceding/succeeding layers, we can also consider performing LAP on the basis of already-pruned layers. This observation brings up a question of the order of pruning; an option is to prune in a forward direction, i.e., prune the preceding layer first and use the pruned weight to prune the succeeding, and the other is to prune backward. Both methods are tested, which are referred to as LAP-forward and LAP-backward, respectively.

Sequential pruning. We also consider a sequential version of LAP-forward/backward methods. More specifically, if we aim to prune total $p\%$ of weights from each layer, we divide the pruning budget into five pruning steps and gradually prune $p/5\%$ of the weights per step in forward/backward direction. Sequential variants will be marked with a suffix -seq.

To deploy lookahead pruning in deeper networks, one must take into account two de facto standard techniques to stabilize the training of deep neural networks, namely, batch normalization (Ioffe & Szegedy, 2015) and residual connections (He et al., 2016).

2.4 Lookahead pruning with batch normalization

Batch normalization (BN), introduced by Ioffe & Szegedy (2015), aims to normalize the output of a layer per batch by scaling and shifting the outputs with trainable parameters. Based on our functional approximation perspective, having batch normalization layers in a neural network is not an issue for MP, which relies on the magnitudes of weights; batch normalization only affects the distribution of the input for each layer, not the layer itself. On the other hand, as the lookahead distortion (Eq. (3)) characterizes the distortion of the multi-layer block, one must take into account batch normalization when assessing the abstract importance of each connection.

The revision of lookahead pruning under the presence of batch normalization can be done fairly simply. Note that such a normalization process can be expressed as

$$x \mapsto a \odot x + b, \tag{8}$$

for some $a, b \in \mathbb{R}^{\mathsf{dim}(x)}$. Hence, we can revise the lookahead pruning to prune the connections with a minimum value of

$$\mathcal{L}_i(w) = |w| \cdot a_{i-1}[j] a_i[k] \cdot \left\| W_{i-1}[j,:] \right\|_F \cdot \left\| W_{i+1}[:,k] \right\|_F, \tag{9}$$

where $a_i[k]$ denotes the k-th index scaling factor for the BN layer placed at the output of the i-th fully-connected or convolutional layer (if BN layer does not exist, let $a_i[k] = 1$). This modification of LAP makes it an efficient pruning strategy, as will be empirically verified in Section 3.2.

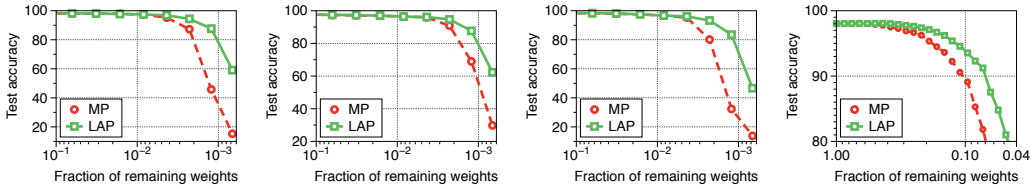
3 Experiments

In this section, we compare the empirical performance of LAP with that of MP. More specifically, we validate the applicability of LAP to nonlinear activation functions in Section 3.1. In Section 3.2, we test the variants of LAP introduced in Section 2.3. In Section 3.3, we deploy LAP on VGG models (Simonyan & Zisserman, 2015) and residual networks (He et al., 2016).

Experiment setup. We consider four neural network architectures: (1) The fully-connected network (FCN) under consideration is composed of four hidden layers, each with 500 hidden neurons. (2) The convolutional network (Conv-6) consists of six convolutional layers, followed by a fully-connected classifier with two hidden layers with 256 hidden neurons each; this model is identical to that appearing in the work of Frankle & Carbin (2019) suggested as a scaled-down variant of VGG.[2] (3) VGGs of depths $\{11, 16, 19\}$ were used, with an addition of batch normalization layers after each convolutional layers, and a reduced number of fully-connected layers from three to one.[3]

[2] Convolutional layers are organized as $[64, 64] - \mathsf{MaxPool} - [128, 128] - \mathsf{MaxPool} - [256, 256]$.

[3] This is a popular configuration of VGG for CIFAR-10 (Liu et al., 2019; Frankle & Carbin, 2019)



(a) sigmoid (b) tanh (c) ReLU (d) before retraining

Figure 3: Test accuracy of FCN with (a) sigmoid, (b) tanh, (c) ReLU activations; (d) Test accuracy of FCN with ReLU activation before retraining, for MNIST dataset.

(4) ResNet-18 was used for the validations with residual connections. All networks are initialized via the method of Glorot & Bengio (2010), and used the ReLU activation function except for the experiments in Section 3.1. We evaluate LAP on image classification tasks. In particular, FCN is trained with MNIST dataset (Lecun et al., 1998), while Conv-6, VGG, and ResNet are trained with CIFAR-10 dataset (Krizhevsky & Hinton, 2009). We focus on the one-shot pruning of MP and LAP, i.e., models are trained with a single training-pruning-retraining cycle. All results in this section are averaged over five independent trials. We present further detailed experimental setup in Appendix C.

3.1 Networks with nonlinear activation functions

We first compare the performance of LAP with that of MP on FCN using three different types of activation functions: sigmoid, and tanh, and ReLU. Figs. 3a to 3c depict the performance of models pruned with LAP (Green) and MP (Red) under various levels of sparsity.

Although LAP was motivated primarily from linear networks and partially justified for positive-homogenous activation functions such as ReLU, the experimental results show that LAP consistently outperforms MP even on networks using sigmoidal activation functions. We remark that LAP outperforms MP by a larger margin as fewer weights survive (less than 1%). Such a pattern will be observed repeatedly in the remaining experiments of this paper.

In addition, we also check whether LAP still exhibits better test accuracy before retraining under the usage of nonlinear activation functions, as in the linear network case (Fig. 2b). Fig. 3d illustrates the test accuracy of pruned FCN using ReLU on MNIST dataset before retraining. We observe that the network pruned by LAP continues to perform better than MP in this case; the network pruned by LAP retains the original test accuracy until only 38% of the weights survive, and shows less than 1% performance drop with only 20% of the weights remaining. On the other hand, MP requires 54% and 30% to achieve the same level of performance, respectively. In other words, the models pruned with MP requires about 50% more survived parameters than the models pruned with LAP to achieve a similar level of performance before being retrained using additional training batches.

3.2 Evaluating LAP variants

Now we evaluate LAP and its variants introduced in Section 2.3 on FCN and Conv-6, each trained on MNIST and CIFAR-10, respectively. Table 1 summarizes the experimental results on FCN and Table 2 summarizes the results on Conv-6. In addition to the baseline comparison with MP, we also compare with random pruning (RP), where the connection to be pruned was decided completely independently. We observe that LAP performs consistently better than MP and RP with similar or smaller variance in any case. In the case of an extreme sparsity, LAP enjoys a significant performance gain; over 75% gain on FCN and 14% on Conv-6. This performance gain comes from a better training accuracy, instead of a better generalization; see Appendix B for more information.

Comparing mono-directional lookahead variants, we observe that LFP performs better than LBP in the low-sparsity regime, while LBP performs better in the high-sparsity regime; in any case, LAP performed better than both methods. Intriguingly, the same pattern appeared in the case of the ordered pruning. Here, LAP-forward can be considered an analogue of LBP in the sense that they both consider layers closer to the input to be more important. Likewise, LAP-backward can be considered an analogue of LFP. We observe that LAP-forward performs better than LAP-backward in the high-sparsity regime, and vice versa in the low-sparsity regime.

Table 1: Test error rates of FCN on MNIST. Subscripts denote standard deviations, and bracketed numbers denote relative gains with respect to MP. Unpruned models achieve 1.98% error rate.

	6.36%	3.21%	1.63%	0.84%	0.43%	0.23%	0.12%
MP (baseline)	1.75±0.11	2.11±0.14	2.53±0.09	3.32±0.27	4.77±0.22	19.85±8.67	67.62±9.91
RP	2.36±0.13	2.72±0.16	3.64±0.17	17.54±7.07	82.48±4.03	88.65±0.00	88.65±0.00
LFP	1.63±0.08	1.89±0.11	2.43±0.10	3.32±0.13	4.23±0.38	9.59±1.70	50.11±12.99
	(-6.41%)	(-10.60%)	(-3.95%)	(-0.12%)	(-11.40%)	(-51.70%)	(-25.91%)
LBP	1.75±0.17	2.04±0.12	2.61±0.15	3.62±0.17	4.19±0.31	9.09±1.41	28.51±14.85
	(+0.69%)	(-3.31%)	(+3.00%)	(+8.97%)	(-12.23%)	(-54.21%)	(-57.84%)
LAP	1.67±0.11	1.89±0.12	2.48±0.13	3.29±0.06	3.93±0.26	6.72±0.44	16.45±5.61
	(-4.24%)	(-10.61%)	(-2.05%)	(-1.08%)	(-17.72%)	(-66.15%)	(-75.68%)
LAP-forward	1.60±0.08	1.93±0.15	2.51±0.11	3.56±0.19	4.47±0.20	6.58±0.33	12.00±0.73
	(-8.25%)	(-8.43%)	(-0.95%)	(+7.03%)	(-6.41%)	(-66.81%)	(-82.26%)
LAP-backward	1.63±0.11	1.88±0.07	2.35±0.02	3.12±0.08	3.87±0.18	5.62±0.17	13.00±3.30
	(-6.64%)	(-10.80%)	(-7.03%)	(-6.08%)	(-19.02%)	(-71.71%)	(-80.78%)
LAP-forward-seq	1.68±0.11	1.92±0.10	2.49±0.14	3.39±0.24	4.21±0.06	6.20±0.32	**10.98±1.03**
	(-3.66%)	(-9.09%)	(-1.42%)	(+1.93%)	(-11.86%)	(-68.73%)	**(-83.76%)**
LAP-backward-seq	**1.57±0.08**	**1.84±0.10**	**2.20±0.10**	**3.13±0.16**	**3.62±0.14**	**5.42±0.27**	11.92±4.61
	(-10.08%)	**(-12.41%)**	**(-13.27%)**	**(-5.90%)**	**(-24.13%)**	**(-72.71%)**	(-82.36%)

Table 2: Test error rates of Conv-6 for CIFAR-10. Subscripts denote standard deviations, and bracketed numbers denote relative gains with respect to MP. Unpruned models achieve 11.97% error rate.

	10.62%	8.86%	7.39%	6.18%	5.17%	4.32%	3.62%
MP (baseline)	11.86±0.33	12.20±0.21	13.30±0.30	15.81±0.59	20.19±2.35	24.43±1.48	28.60±2.10
RP	26.85±1.23	29.72±1.13	32.98±1.10	35.92±1.08	39.13±1.05	41.20±1.19	43.60±0.82
LFP	11.81±0.35	12.18±0.23	13.27±0.44	15.04±0.43	18.50±0.80	22.86±1.66	26.65±1.33
	(-0.39%)	(-0.20%)	(-0.26%)	(-4.87%)	(-8.37%)	(-6.40%)	(-6.83%)
LBP	12.08±0.17	12.34±0.36	13.26±0.16	14.93±0.85	18.11±1.27	22.57±0.94	26.34±1.60
	(+1.84%)	(-1.15%)	(-0.33%)	(-5.57%)	(-10.31%)	(-7.59%)	(-7.91%)
LAP	**11.76±0.24**	**12.16±0.27**	**13.05±0.14**	14.39±0.44	17.10±1.26	21.24±1.16	24.52±1.11
	(-0.83%)	**(-0.34%)**	**(-1.86%)**	(-8.99%)	(-15.30%)	(-13.04%)	(-14.29%)
LAP-forward	11.82±0.16	12.35±0.34	13.09±0.36	14.42±0.45	17.05±1.30	20.28±1.40	22.80±0.51
	(-0.33%)	(+1.24%)	(-1.62%)	(-8.79%)	(-15.57%)	(-16.98%)	(-20.30%)
LAP-backward	11.82±0.25	12.29±0.06	12.93±0.38	14.55±0.58	17.00±0.84	20.00±0.82	23.37±1.16
	(-0.32%)	(+0.68%)	(-2.78%)	(-7.98%)	(-15.78%)	(-18.11%)	(-18.30%)
LAP-forward-seq	12.01±0.17	12.47±0.37	13.19±0.19	**14.12±0.28**	**16.73±0.95**	**19.63±1.81**	**22.44±1.31**
	(+1.28%)	(+2.21%)	(-0.81%)	**(-10.70%)**	**(-17.13%)**	**(-19.62%)**	**(-21.54%)**
LAP-backward-seq	11.81±0.16	12.35±0.26	13.25±0.21	14.17±0.44	16.99±0.97	19.94±1.02	23.15±1.12
	(-0.39%)	(+1.25%)	(-0.41%)	(-10.37%)	(-15.87%)	(-18.38%)	(-19.08%)

Finally, we observe that employing forward/backward ordering tricks and sequential methods leads to better performance, especially in the high-sparsity regime. There are no clear benefits of adopting a directional method in the low-sparsity regime; while the performance was generally better than default LAP, there seems to be an issue with the stability of both methods. In FCN, for instance, we observe that LAP-forward(-seq) attains a higher test error rate than MP when 0.84% of the weights remain unpruned. In Conv-6, both LAP-forward/backward(-seq) performed worse than MP at the sparsity level where 8.86% fraction of weights survive.

3.3 Deeper networks: VGG and ResNet

We also compare the empirical performances of MP with LAP on VGG-$\{11, 16, 19\}$ (Tables 3 to 5, respectively) and ResNet-18 (Table 6) trained on CIFAR-10 dataset. We perform additional experiments on LAP-forward to verify the observation that it outperforms LAP in the high-sparsity regime, on such deeper network architectures. As in the last section, we observe that LAP outperforms MP consistently at all sparsity levels in VGG. In particular, test accuracies decay at a much slower rate with LAP. Most notably, we observe that the models pruned by LAP retain test accuracies of 70$\sim$80% even with less than 2% of the weights remaining. In contrast, the performance of models pruned with MP falls drastically, to below 50% accuracy.

Table 3: Test error rates for VGG-11. Subscripts denote standard deviations, and bracketed numbers denote relative gains with respect to MP. Unpruned models achieve 11.51% error rate.

	16.74%	12.10%	8.74%	6.32%	4.56%	3.30%	2.38%	1.72%
MP (baseline)	11.41±0.24	12.38±0.14	13.54±0.35	16.08±1.13	19.76±1.67	28.12±3.45	45.38±11.69	55.97±15.99
LAP	**11.19±0.15**	**11.79±0.44**	**12.95±0.14**	**13.95±0.17**	15.59±0.35	20.96±6.02	22.00±1.09	28.96±3.30
	(-1.96%)	**(-4.78%)**	**(-4.39%)**	**(-13.25%)**	(-21.13%)	(-25.47%)	(-51.52%)	(-48.25%)
LAP-forward	11.47±0.30	12.33±0.12	13.15±0.22	13.96±0.25	**15.42±0.21**	**18.22±0.69**	**21.74±1.59**	**25.85±1.40**
	(+0.56%)	(-0.44%)	(-2.87%)	(-13.18%)	**(-21.97%)**	**(-35.20%)**	**(-52.10%)**	**(-53.82%)**

Table 4: Test error rates for VGG-16. Subscripts denote standard deviations, and bracketed numbers denote relative gains with respect to MP. Unpruned models achieve 9.33% error rate.

	10.28%	7.43%	5.37%	3.88%	2.80%	2.03%	1.46%	1.06%
MP (baseline)	9.55±0.11	10.78±0.45	13.42±2.19	17.83±3.08	26.61±4.91	48.87±5.85	69.39±11.85	83.47±5.60
LAP	**9.35±0.18**	**10.07±0.19**	**11.52±0.26**	**12.57±0.34**	**14.23±0.27**	**17.01±1.46**	25.03±2.08	32.45±12.20
	(-2.05%)	**(-6.59%)**	**(-14.21%)**	**(-29.50%)**	**(-46.52%)**	**(-65.19%)**	(-63.92%)	(-61.12%)
LAP-forward	9.45±0.17	10.40±0.20	11.33±0.15	13.09±0.21	14.61±0.25	17.10±0.19	**22.39±0.74**	**24.99±0.49**
	(-1.03%)	(-3.49%)	(-15.60%)	(-26.56%)	(-45.08%)	(-65.02%)	**(-67.74%)**	**(-70.06%)**

Table 5: Test error rates for VGG-19. Subscripts denote standard deviations, and bracketed numbers denote relative gains with respect to MP. Unpruned models achieve 9.02% error rate.

	12.09%	8.74%	6.31%	4.56%	3.30%	2.38%	1.72%	1.24%
MP (baseline)	8.99±0.12	9.90±0.09	11.43±0.24	15.62±1.68	29.10±8.78	40.27±11.51	63.27±11.91	77.90±7.94
LAP	**8.89±0.14**	**9.51±0.22**	**10.56±0.28**	**12.11±0.44**	13.64±0.77	16.38±1.47	20.88±1.71	22.82±0.81
	(-1.07%)	**(-3.96%)**	**(-7.63%)**	**(-22.48%)**	(-53.13%)	(-59.31%)	(-67.00%)	(-70.71%)
LAP-forward	9.63±0.25	10.31±0.23	11.10±0.22	12.24±0.33	**13.54±0.28**	**16.03±0.46**	**19.33±1.14**	**21.59±0.32**
	(+7.16%)	(+4.12%)	(-2.89%)	(-21.66%)	**(-53.46%)**	**(-60.18%)**	**(-69.44%)**	**(-72.29%)**

Table 6: Test error rates for ResNet-18. Subscripts denote standard deviations, and bracketed numbers denote relative gains with respect to MP. Unpruned models achieve 8.68% error rate.

	10.30%	6.33%	3.89%	2.40%	1.48%	0.92%	0.57%	0.36%
MP (baseline)	8.18±0.33	**8.74±0.15**	9.82±0.18	**11.28±0.30**	14.31±0.18	18.56±0.36	22.93±0.93	26.77±1.04
LAP	**8.09±0.10**	8.97±0.22	**9.74±0.15**	11.35±0.20	**13.73±0.24**	**16.29±0.29**	**20.22±0.53**	**22.45±0.64**
	(-1.08%)	(+2.59%)	**(-0.81%)**	(+0.64%)	**(-4.08%)**	**(-12.23%)**	**(-11.82%)**	**(-15.82%)**
LAP-forward	8.19±0.15	9.17±0.07	10.32±0.27	12.38±0.30	15.31±0.62	18.56±0.88	21.09±0.53	23.89±0.46
	(+0.12%)	(+4.85%)	(+5.09%)	(+9.79%)	(+6.96%)	(-0.02%)	(-8.04%)	(-10.44%)

In ResNet-18 where we deploy LAP without an explicit mechanism to handle residual connections, we still observe performance gains in most cases, but the gain is not significant until the sparsity gets extremely low. Note that our LAP-pruned models maintain a smaller variance in test error rates compared to MP-pruned models. In all experiments, we find that LAP requires approximately 50% fewer parameters compared to MP, to achieve a similar error rate in the high-sparsity regime.

Again, the ordered pruning method (LAP-forward) outperforms both LAP and MP in the high-sparsity regime (more than 95% weights pruned). On the other hand, LAP-forward is also consistently worse (by at most 1%) than MP or LAP in the low-sparsity regime. In the case of ResNet-18, LAP performs better than LAP-forward for every sparsity levels tested. LAP-forward still performs better than MP in the high-sparsity regime. We think LAP-forward underperforms LAP because we ignore residual connections in computing its Frobenius distortion. Handling residual connections in LAP variants is an interesting direction to explore in the future.

4 Conclusion

In this work, we interpret magnitude-based pruning as a solution to the minimization of the Frobenius distortion of a single layer operation incurred by pruning. Based on this framework, we consider the minimization of the Frobenius distortion of multi-layer operation, and propose a novel lookahead pruning (LAP) scheme as a computationally efficient algorithm to solve the optimization. Although LAP was motivated from linear networks, we empirically show its effectiveness on networks with nonlinear activation functions, and test the algorithm on various network architectures including VGG and ResNet, where LAP consistently performs better than MP.

REFERENCES

S. Arora, N. Cohen, and E. Hazan. On the optimization of deep networks: Implicit acceleration by overparametrization. In *Proceedings of the International Conference on Machine Learning*, 2018.

G. Bellec, D. Kappel, W. Maass, and R. Legenstein. Deep rewiring: training very sparse deep networks. In *International Conference on Learning Representations*, 2018.

B. Dai, C. Zhu, B. Guo, and D. Wipf. Compressing neural networks using the variational information bottleneck. In *Proceedings of the International Conference on Machine Learning*, 2018.

X. Ding, G. Ding, Y. Guo, and J. Han. Centripetal SGD for pruning very deep convolutionnal networks with complicated structure. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

S. S. Du, W. Hu, and J. D. Lee. Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced. In *Advances in Neural Information Processing Systems*, 2018.

J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. arXiv preprint 1902.09574, 2019.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2010.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, 2015.

B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, 1993.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Y. He, J. Lin, Z. Liu, H. Wang, L. Li, and S. Han. AMC: AutoML for model compression and acceleration on mobile devices. In *European Conference on Computer Vision*, 2018.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*, 2015.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1989.

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

J. Lin, Y. Rao, J. Lu, and J. Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, 2017.

Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision*, 2017.

Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

C. Louizos, K. Ullrich, and M. Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, 2017.

C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsified deep neural networks. In *Proceedings of the International Conference on Machine Learning*, 2017.

K. G. Murty and S. N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical programming*, 39(2):117–129, 1987.

R. M. Neal. *Bayesian learning for neural networks*. Springer, 1996.

H. Sedghi, V. Gupta, and P. M. Long. The singular values of convolutional layers. In *International Conference on Learning Representations*, 2019.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep networks. In *Advances in Neural Information Processing Systems*, 2016.

D. Zhang, H. Wang, M. Figueiredo, and L. Balzano. Learning to share: simultaneous parameter tying and sparsification in deep learning. In *International Conference on Learning Representations*, 2018.

M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

A ON NP-HARDNESS OF EQ. (3)

In this section, we show that the optimization in Eq. (3) is NP-hard by showing the reduction from the following binary quadratic programming which is NP-hard (Murty & Kabadi, 1987):

$$\min_{x \in \{0,1\}^n} x^T A x \tag{10}$$

for some symmetric matrix $A \in \mathbb{R}^{n \times n}$. Without loss of generality, we assume that the minimum eigenvalue of A (denoted with λ) is negative; if not, Eq. (10) admits a trivial solution $x = (0, \ldots, 0)$.

Assuming $\lambda < 0$, Eq. (10) can be reformulated as:

$$\min_{x \in \{0,1\}^n} x^T H x + \lambda \sum_i x_i \tag{11}$$

where $H = A - \lambda I$. Here, one can easily observe that the above optimization can be solved by solving the below optimization for $s = 1, \ldots, n$

$$\min_{x \in \{0,1\}^n : \sum_i x_i = s} x^T H x \tag{12}$$

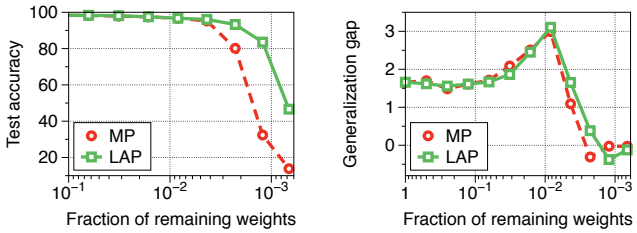
Finally, we introduce the below equality

$$\begin{aligned} x^\top H x &= x^\top U \Lambda U^\top x && (13)\\ &= \|\sqrt{\Lambda} U^\top x\|_F^2 && (14)\\ &= \|\sqrt{\Lambda} U^\top x\|_F^2 && (15)\\ &= \|\sqrt{\Lambda} U^\top \mathbf{1} - \sqrt{\Lambda} U^\top \big((\mathbf{1} - x) \odot \mathbf{1}\big)\|_F^2 && (16) \end{aligned}$$

where $\mathbf{1}$ denotes a vector of ones, U is a matrix consisting of the eigenvectors of H as its column vectors, and Λ is a diagonal matrix with corresponding (positive) eigenvalues of H as its diagonal elements. The above equality shows that Eq. (12) is a special case of Eq. (3) by choosing $W_1 = \sqrt{\Lambda} U^\top, W_2 = \mathbf{1}, W_3 = 1$ and $M = \mathbf{1} - x$. This completes the reduction from Eq. (10) to Eq. (3).

B WHERE IS THE PERFORMANCE GAIN OF LAP COMING FROM?

In this section, we briefly discuss where the benefits of the sub-network discovered by LAP comes from; does LAP subnetwork have a better generalizability or expressibility? For this purpose, we look into the generalization gap, i.e., the gap between the training and test accuracies, of the hypothesis learned via LAP procedure. Below we present a plot of test accuracies (Fig. 4a) and a plot of generalization gap (Fig. 4b) for FCN trained with MNIST dataset. The plot hints us that the network structure learned by LAP may not necessarily have a smaller generalizability. Remarkably, the generalization gap of the MP-pruned models and the LAP-pruned models are very similar to each other; the benefits of LAP subnetwork compared to MP would be that it can express a better-performing architecture with a network of similar sparsity and generalizability.



(a) Test accuracy (b) Generalization gap

Figure 4: Test accuracy and generalization gap of FCN trained on MNIST.

C EXPERIMENTAL SETUP

Models and datasets. We consider four neural network architectures: (1) The fully-connected network (FCN) under consideration is composed of four hidden layers, each with 500 hidden neurons. (2) The convolutional network (Conv-6) consists of six convolutional layers, followed by a fully-connected classifier with two hidden layers with 256 hidden neurons each; this model is identical to that appearing in the work of Frankle & Carbin (2019) suggested as a scaled-down variant of VGG.[4] (3) VGGs of depths $\{11, 16, 19\}$ were used, with an addition of batch normalization layers after each convolutional layers, and a reduced number of fully-connected layers from three to one.[5] (4) ResNet-18 was used for the validations with residual connections. All networks were initialized via the method of Glorot & Bengio (2010), and used the ReLU activation function except for the experiments in Section 3.1. We focus on image classification tasks. FCN is trained with MNIST dataset (Lecun et al., 1998), while Conv-6, VGG, and ResNet are trained with CIFAR-10 dataset (Krizhevsky & Hinton, 2009).

Optimizers and hyperparameters. We use Adam optimizer (Kingma & Ba, 2015) with batch size 60. We use a learning rate of $1.2 \cdot 10^{-3}$ for FCN and $3 \cdot 10^{-4}$ for all other models. For FCN, we use [50k,50k] for the initial training phase and retraining phase. For Conv-6, we use [30k,20k] steps. For VGG-11 and ResNet-18, we use [35k,25k] steps. For VGG-16, we use [50k,35k]. For VGG-19, we use [60k,40k]. We do not use any weight decay, learning rate scheduling, or regularization.

Sparsity levels. To determine the layerwise pruning ratio, we largely follow the the guidelines of Han et al. (2015); Frankle & Carbin (2019): For integer values of τ, we keep p^τ fraction of weights in all convolutional layers and q^τ fraction in all fully-connected layers, except for the last layer where we use $(1+q)/2$ instead. For FCN, we use $(p, q) = (0, 0.5)$. For Conv-6, VGGs and ResNet-18, we use $(0.85, 0.8)$. For ResNet-18, we do not prune the first convolutional layer. The range of sparsity for reported figures in Tables 1 to 6 is decided as follows: we start from τ where test error rate starts falling below that of an unpruned model and report the results at $\tau, \tau + 1, \tau + 2, \ldots$ for FCN and Conv-6, $\tau, \tau + 2, \tau + 4, \ldots$ for VGGs and $\tau, \tau + 3, \tau + 6, \ldots$ for ResNet-18.

[4] Convolutional layers are organized as $[64, 64] - \mathsf{MaxPool} - [128, 128] - \mathsf{MaxPool} - [256, 256]$.

[5] This is a popular configuration of VGG for CIFAR-10 (Liu et al., 2019; Frankle & Carbin, 2019)