Neural Network Compression via Additive Combination of Reshaped, Low-rank Matrices

Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán

Department of Computer Science and Engineering
University of California, Merced
{yidelbayev, mcarreira-perpinan}@ucmerced.edu

Abstract

In the last five years, neural network compression has become an important problem due to the increasing necessity of running complex networks on small devices. We consider a form of network compression that has not been explored before: an additive combination of reshaped low-rank matrices. That is, given the weights of a neural network, we constrain them as a sum of differently shaped low-rank matrices to reduce the network's size and inference demands. Computationally, this is a hard problem involving integer variables (ranks) and continuous variables (weights), as well as nonlinear loss and constraints. We formulate it as a model selection over the family of compressed models and give an optimization algorithm that efficiently handles the inherent combinatorial structure. This results in a "Learning-Compression" algorithm which alternates between a standard machine learning step and a step involving signal compression. We demonstrate the effectiveness of the proposed compression scheme and the corresponding algorithm on multiple networks and datasets.

1 Introduction

With state-of-the-art results across multiple machine learning problems, deep neural networks have become a standard computational tool for practical applications involving image and video recognition, speech understanding, signal enhancement, and many others. This improvement in the performance comes at the cost of the ever-increasing hardware demands, which motivates the problem of *model compression*: given a trained network with particular task performance (e.g. classification accuracy), how can we reduce its size in terms of required memory, compute, energy, and other costs of interest while minimally affecting the performance of the model?

While many compression strategies have been developed in the literature, we focus on using the low-rank decompositions. Such a compression scheme has several advantages. Firstly, low-rank methods have a history of usage in the fields of linear algebra, signal processing, and statistics with robust computational routines like singular value decomposition (SVD) and well-tested software packages like BLAS and LAPACK. Secondly, when the neural network weights are compressed using the matrices with appropriately small ranks, it *reduces both the size of the network and the computational requirements needed for the forward pass.* Most importantly, the computational savings are realizable without explicit support from the hardware. Indeed, if the weight matrix $\mathbf{W}$ is of low rank, it can be seen as a product of matrices $\mathbf{U}\mathbf{V}^T$. The forward pass $\mathbf{W}\mathbf{x}$ through such layer now can be computed as a forward pass

through a sequence of two regular layers: first through a layer with weights $\mathbf{V}^T$ and then through a layer with weights $\mathbf{U}$. This hardware friendliness is in stark contrast compared to other compression schemes, e.g. quantized or elementwise pruned models require building a dedicated processor to be efficiently deployed [1].

While previous methods use a single low-rank matrix to compress the original weights $\mathbf{W}$, we propose to use an additive combination of the form $\mathbf{W} = \mathbf{\Theta}_1 + \mathbf{\Theta}_2$ where each additive term is of low rank. Without special treatment, such a scheme has a trivial effect: the sum of two matrices of rank r_1 and r_2 can always be parameterized as another low-rank matrix with rank $r_3 \leq r_1 + r_2$.

One particular way to circumvent the limitations of naive additive low rank is by imposing the low-rank constraint on the reshaped versions of the matrices. Formally, assume we have a weight matrix $\mathbf{\Theta}$ of shape $m \times n$ (same as $\mathbf{W}$). Let us define a *reshape* $\mathcal{R}(\mathbf{\Theta})$ to be a re-ordering of the nm elements of $\mathbf{\Theta}$ into some $p \times q$ sized matrix (with $qp = mn$), such that $\mathcal{R}(\mathbf{\Theta})$ contains the same set of elements as $\mathbf{\Theta}$, but in different order[1]. Using this definition of reshape, we can write the additive low-rank scheme as follows:

$$\mathbf{W} = \mathbf{\Theta}_1 + \mathbf{\Theta}_2, \quad \text{rank}\left(\mathcal{R}_1(\mathbf{\Theta}_1)\right) = r_1, \quad \text{rank}\left(\mathcal{R}_2(\mathbf{\Theta}_2)\right) = r_2. \tag{1}$$

Here, $\mathcal{R}_1$ and $\mathcal{R}_2$ are two different reshaping functions chosen accordingly (sec. 1.1).

The additive low-rank scheme given by eq. (1) has several advantages. First, it includes regular low rank as a special case: if we set $\mathcal{R}_1$ to be an identity reshape and set the rank $r_2 = 0$ we recover regular low-rank constraint on $\mathbf{W}$. Second, if the scheme is applied separately per layer, the forward pass through such layer can be computed efficiently and in parallel. Indeed, instead of computing $\mathbf{Wx}$, we compute in parallel the results of $\mathbf{\Theta}_1\mathbf{x}$ and $\mathbf{\Theta}_2\mathbf{x}$ and then sum the final outputs. Under proper reshaping choices (sec. 1.1), the forward pass through $\mathbf{\Theta}_1\mathbf{x}$ and $\mathbf{\Theta}_2\mathbf{x}$ can be further sped up using the factorization of weights due to low-rank structure, and these speed-ups do not require special hardware support to be materialized. The actual amount of the speedup and the reduction of storage depends on the ranks r_1 and r_2.

Given our compression scheme (1) defined for each layer, an important remaining question is, *how to apply it to the neural network compression problem?* We would like to achieve the best possible compression under a certain cost function (e.g. inference speed) while maintaining the network accuracy as close as possible to the *uncompressed reference model*. In principle, we can try different values of (r_1, r_2)-pairs per layer, decompose the weight matrices according to the (r_1, r_2)-induced additive scheme, retrain all decomposed weights on the network's task using backpropagation, and measure the network's accuracy and the compression cost function. By sampling many combinations of ranks, we can trace the entire compression space and find the best model under the given compression cost. Unfortunately, this approach does not scale well. There is a combinatorial number of configurations of ranks that need to be checked, and every check requires full (re)training of the neural network.

To handle the combinatorial number of rank configurations, we will define an optimization problem to jointly learn the weights and ranks of our additive scheme that

[1]This operation is similar to the Matlab's `reshape` function, `https://www.mathworks.com/help/matlab/ref/reshape.html`.

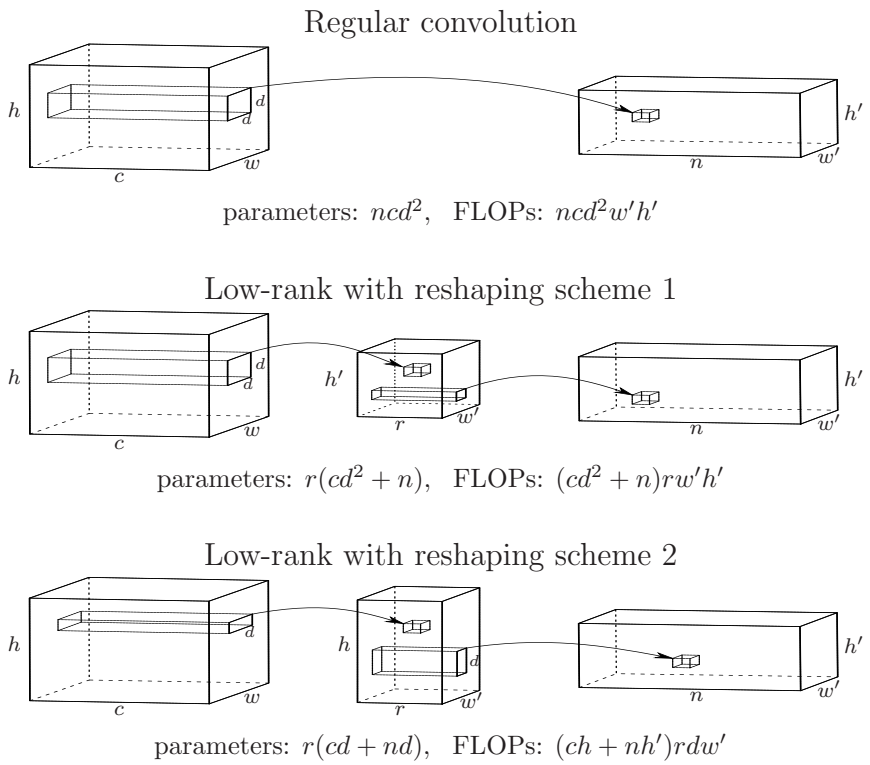

Figure 1: A graphical illustration of the inference (computation of $\mathbf{Wx}$) through a regular convolutional layer (top) and the r-rank versions when using different reshapes on the weights (scheme 1 and 2). While the layer's input and output remain unchanged (input: tensor of size $h \times c \times w$, output: tensor of size $h' \times n \times w'$), the actual floating-point operations (FLOPs) required to do the forward pass and the total number of the weights are different. When using our additive combination, the input will be mapped using both schemes 1 and 2 (with separately learned ranks per scheme) and then combined using elementwise addition.

will select the best-suited configuration under a compression cost function. We give the formulation of the problem in section 2, and in section 3 we give an alternating optimization algorithm that efficiently handles the combinatorial nature of the problem. In section 4, we validate the proposed scheme and its optimization algorithm on various networks. In the remainder of this section, we fill in the details of matrix reshapes used with our additive scheme and give a brief overview of related work.

1.1 The choice of the reshapes

While the additive combination of reshaped low-rank matrices can be applied to any layer, we limit our attention to the application of our scheme to the compression of convolutional layers. Interestingly, in such setting, the notion of the weight reshape arises naturally.

The weights of the convolutional layer are typically stored in a tensor of shape $n \times c \times d \times d$: here n is the number of filters, c is the number of channels and $d \times d$ is spatial resolution. To apply the low-rank constraints, we need to reshape this tensor into a matrix, for which two practical reshaping schemes were proposed in the

literature. The scheme 1 reshapes the weights into an $n \times cd^2$ matrix by stacking $c \times d \times d$ filters as columns [2, 3]; the scheme 2 reshapes the weights into an $nd \times cd$ matrix by stacking the filters across the spatial dimension [4, 5]. The advantage of these reshapes lies in the implementation of the forward pass: when the reshaped tensor is of low rank, the forward pass through such layer can be implemented as a sequence of two convolutional layers (see Fig. 1).

1.2 Related work

Low-rank schemes have been used to compress neural networks trained on various tasks. Early methods [2, 3, 6] used a compress-and-retrain approach where weight matrices were converted to low-rank using singular value thresholding. This step involves estimation of the ranks through heuristics or other side routines and completely disregards the model's original task (e.g. classification cross-entropy). Therefore, after the compression is finished, these methods perform a fine-tuning of the network on its original loss. In contrast, recent methods improved the low-rank compression by jointly training the weights and the ranks to suit the compression goals [7, 8].

2 Problem formulation

We formulate the problem as *model compression as constrained optimization* [9] and give a *Learning-Compression* (LC) algorithm [8–14] to solve it. We will show that such a compression formulation naturally expresses our desired structure over the model weights, and results in a very convenient algorithm.

Assume we are given a K-layer neural network with weights $\mathbf{W} = \{\mathbf{W}^1, \dots, \mathbf{W}^K\}$ where $\mathbf{W}^k$ is the weight matrix (or tensor) of the layer k. Additionally, we assume that the network is well trained on some task (e.g. classification) by minimizing the loss $L(\mathbf{W})$. We introduce the additive low-rank scheme of eq. (1) for each layer of the neural network as an explicit constraint, and include the ranks and the parameters into the optimization. We control the compression-accuracy tradeoff using the *compression cost function* $C(\boldsymbol{\Theta}, \mathbf{r})$, and define the optimization objective as a λ-weighted sum ($\lambda > 0$) of cost C and loss L subject to constraints:

$$\begin{aligned} \min_{\mathbf{W},\boldsymbol{\Theta},\mathbf{r}} \quad & L(\mathbf{W}) + \lambda\, C(\boldsymbol{\Theta}, \mathbf{r}) \\ \text{s.t.} \quad & \mathbf{W}^k = \boldsymbol{\Theta}_1^k + \boldsymbol{\Theta}_2^k, \quad \text{for} \quad k = 1 \dots K, \\ & \operatorname{rank}\left(\mathcal{R}_1(\boldsymbol{\Theta}_1^k)\right) = r_1^k, \quad \operatorname{rank}\left(\mathcal{R}_2(\boldsymbol{\Theta}_2^k)\right) = r_2^k. \end{aligned} \tag{2}$$

Here, $\boldsymbol{\Theta}_1^k$ and $\boldsymbol{\Theta}_2^k$ are additive terms that form the weights of the layer k, and r_1^k and r_2^k are the respective ranks imposed on the reshaped $\boldsymbol{\Theta}_1^k$ and $\boldsymbol{\Theta}_2^k$. We collectively refer to all additive terms across the layers as $\boldsymbol{\Theta}$, and to all ranks as $\mathbf{r}$:

$$\boldsymbol{\Theta} = \left\{\boldsymbol{\Theta}_1^k, \boldsymbol{\Theta}_2^k\right\}_{k=1}^K \text{ and } \mathbf{r} = \left\{r_1^k, r_2^k\right\}_{k=1}^K.$$

The term $\lambda\, C(\boldsymbol{\Theta}, \mathbf{r})$ controls the amount of compression and is, in general, a function of the weights and ranks throughout the network. For costs of interest, such

as the total number of parameters or FLOPs, C is a layerwise separable function dependent only on the ranks:

$$C(\boldsymbol{\Theta}, \mathbf{r}) = C(\mathbf{r}) = \sum_{k=1}^{K} C_k(r_1^k, r_2^k), \tag{3}$$

where the per-layer cost $C_k(r_1^k, r_2^k)$ is defined as:

$$C_k(r_1^k, r_2^k) = \alpha_1 r_1^k + \alpha_2 r_2^k, \tag{4}$$

for some constants $\alpha_1, \alpha_2 > 0$. Indeed, if we are optimizing for the number of parameters, the total number of compressed parameters of the layer k is the total number of the parameters in $\boldsymbol{\Theta}_k^1$ and $\boldsymbol{\Theta}_k^2$, which is a function of the ranks r_1^k and r_2^k. This also holds for the cost of the total number of FLOPs, see Fig. 1.

3 Optimization algorithm

Our formulation of compression in eq. (2) is a mixed-integer optimization problem involving a nonlinear loss function L (defined by a neural network and a machine learning (ML) task) and complex constraints involving the sum of reshaped matrices for which both weights and ranks need to be learned. Model compression problems of this form (defined as minimizing the sum of model loss and compression cost subject to compression constraints) can be conveniently optimized using the Learning-Compression framework [9]. This results in an algorithm that alternates over a step that has the form of a standard ML problem (network learning) and a step that has the form of a standard signal compression problem involving the squared distortion.

To derive the algorithm, we apply a penalty method [15, ch.17] to the additive constraints in (2). In practice, we use the more effective augmented-Lagrangian method, but here we describe the quadratic-penalty method, which captures the intuition of the LC algorithm and is simpler. We optimize the following while driving $\mu \to \infty$:

$$\begin{aligned} \min_{\mathbf{W}, \boldsymbol{\Theta}, \mathbf{r}} \quad & L(\mathbf{W}) + \lambda\, C(\boldsymbol{\Theta}, \mathbf{r}) + \frac{\mu}{2} \sum_{k=1}^{K} \left\| \mathbf{W}^k - \boldsymbol{\Theta}_1^k - \boldsymbol{\Theta}_2^k \right\|^2 \\ \text{s.t.} \quad & \operatorname{rank}\left(\mathcal{R}_1(\boldsymbol{\Theta}_1^k)\right) = r_1^k, \quad \operatorname{rank}\left(\mathcal{R}_2(\boldsymbol{\Theta}_2^k)\right) = r_2^k, \quad k = 1 \dots K. \end{aligned} \tag{5}$$

where all norms are Frobenius throughout the paper. Now, we can obtain the LC algorithm by alternating over $\mathbf{W}$ and $\{\boldsymbol{\Theta}, \mathbf{r}\}$, which results in the following two steps:

- The step over $\mathbf{W}$, which we call a *learning (L) step*, has the form of:

$$\min_{\mathbf{W}} \quad L(\mathbf{W}) + \frac{\mu}{2} \sum_{k=1}^{K} \left\| \mathbf{W}^k - \boldsymbol{\Theta}_1^k - \boldsymbol{\Theta}_2^k \right\|^2.$$

- The step over $\{\boldsymbol{\Theta}, \mathbf{r}\}$, which we call a *compression (C) step*, has the form of:

$$\begin{aligned} \min_{\boldsymbol{\Theta}, \mathbf{r}} \quad & \lambda\, C(\boldsymbol{\Theta}, \mathbf{r}) + \frac{\mu}{2} \sum_{k=1}^{K} \left\| \mathbf{W}^k - \boldsymbol{\Theta}_1^k - \boldsymbol{\Theta}_2^k \right\|^2 \\ \text{s.t.} \quad & \operatorname{rank}\left(\mathcal{R}_1(\boldsymbol{\Theta}_1^k)\right) = r_1^k, \quad \operatorname{rank}\left(\mathcal{R}_2(\boldsymbol{\Theta}_2^k)\right) = r_2^k, \quad k = 1 \dots K. \end{aligned}$$

Algorithm 1 Quadratic Penalty (QP) version of our algorithm to jointly learn ranks and weights to compress a neural network with additive low-rank scheme.

input K-layer neural net with weights $\mathbf{W} = (\mathbf{W}^1, \dots, \mathbf{W}^K)$,
hyperparameter λ, cost function C, distinct reshaping functions $\mathcal{R}_1$ and $\mathcal{R}_2$

$\mathbf{W} = (\mathbf{W}^1, \dots, \mathbf{W}^K) \leftarrow \arg\min_{\mathbf{W}} L(\mathbf{W})$ reference net

$\mathbf{r} = \left\{r_1^k, r_2^k\right\}_{k=1}^K \leftarrow \mathbf{0}$ ranks

$\mathbf{\Theta} = \left\{\mathbf{\Theta}_1^k, \mathbf{\Theta}_2^k\right\}_{k=1}^K \leftarrow \mathbf{0}$ additive terms

for $\mu = \mu_1 < \mu_2 < \cdots < \mu_T$

$$\mathbf{W} \leftarrow \arg\min_{\mathbf{W}} L(\mathbf{W}) + \frac{\mu}{2} \sum_{k=1}^{K} \|\mathbf{W}^k - \mathbf{\Theta}_1^k - \mathbf{\Theta}_2^k\|^2 \quad \text{L step}$$

for $k = 1, \dots, K$ C step

repeat for S times

$$\mathbf{\Theta}_1^k, r_1^k \leftarrow \arg\min_{\mathbf{\Theta}_1^k, r_1^k} \lambda\, C_k(r_1^k, r_2^k) + \frac{\mu}{2}\left\|\mathbf{W}^k - \mathbf{\Theta}_2^k - \mathbf{\Theta}_1^k\right\|^2 \text{ s.t. } \operatorname{rank}\left(\mathcal{R}_1(\mathbf{\Theta}_1^k)\right) = r_1^k$$

$$\mathbf{\Theta}_2^k, r_2^k \leftarrow \arg\min_{\mathbf{\Theta}_2^k, r_2^k} \lambda\, C_k(r_1^k, r_2^k) + \frac{\mu}{2}\left\|\mathbf{W}^k - \mathbf{\Theta}_1^k - \mathbf{\Theta}_2^k\right\|^2 \text{ s.t. } \operatorname{rank}\left(\mathcal{R}_2(\mathbf{\Theta}_2^k)\right) = r_2^k$$

return $\mathbf{W}, \mathbf{\Theta}, \mathbf{r}$

Let us make a few observations. Our choice of splitting the variables into groups of $\mathbf{W}$ and $\mathbf{\Theta}, \mathbf{r}$ has a significant effect on the steps. The L step has the form of learning a regular neural network with added ℓ_2 penalty. The penalty drives the weights $\mathbf{W}$ towards the additive combination of low-rank matrices, while maintaining the smallest loss L. The L-step problem is a standard deep neural net training problem, where the loss function depends on a large dataset. This optimization can be difficult and costly, but can be conveniently solved with deep learning software (e.g. PyTorch) using hardware accelerators (such as GPUs).

On the other hand, the C-step optimization problem is independent of the task loss L, does not require evaluation over a dataset, and formulates a problem of finding the best configuration of the ranks that fits the weights $\mathbf{W}$ in the ℓ_2 sense. This step is the actual compression of the weights of the neural network using our additive low-rank scheme. We discuss its solution next.

Solution of the C step Due to the layerwise separability of the cost function C (eqs. (3)–(4)), the C-step problem separates over the layers into K smaller problems:

$$\begin{aligned} \min_{\mathbf{\Theta}_1^k, \mathbf{\Theta}_2^k, r_1^k, r_2^k} \quad & \lambda\, C_k(r_1^k, r_2^k) + \frac{\mu}{2}\left\|\mathbf{W}^k - \mathbf{\Theta}_1^k - \mathbf{\Theta}_2^k\right\|^2 \\ \text{s.t.} \quad & \operatorname{rank}\left(\mathcal{R}_1(\mathbf{\Theta}_1^k)\right) = r_1^k, \quad \operatorname{rank}\left(\mathcal{R}_2(\mathbf{\Theta}_2^k)\right) = r_2^k. \end{aligned} \tag{6}$$

We are not aware of any closed-form solution for this problem. However, we can again use alternating optimization, over the groups of $\{\mathbf{\Theta}_1^k, r_1^k\}$ and $\{\mathbf{\Theta}_2^k, r_2^k\}$. In this case each subproblem does have a closed-form solution, involving SVD and selection over the ranks [8]. We do not have a proof that this alternating optimization will always

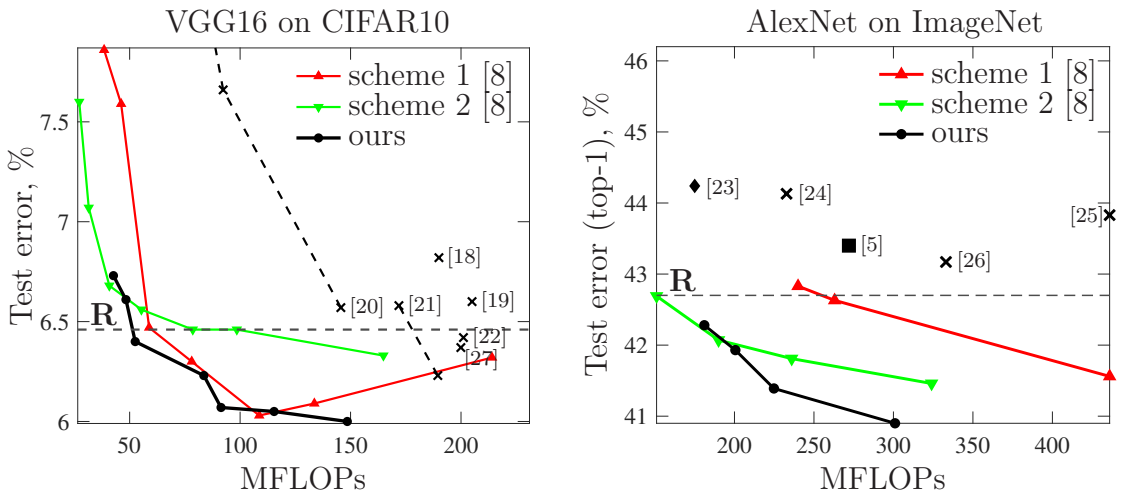

Figure 2: Results of compressing VGG16 (on CIFAR10) and AlexNet (on ImageNet) using individual low-rank schemes 1, 2 [8], and using our additive combination of scheme 1 and 2. Ideally, we would like to have models on the left bottom corner corresponding to more accurate models with fewer FLOPs. We additionally plot (as individual markers) recent results from the literature focused on reducing the FLOPs: ×, pruning [18–22, 24–26]; ⧫, quantization [23]; ■, low-rank [5]. The horizontal dashed lines marked with **R** indicate the test errors of the reference models (whose very large MFLOPs are well outside both plots). For both networks, our compressed results achieve better error-FLOPs tradeoff when compared to either the individual scheme-1 and scheme-2 results or other methods.

find the globally optimum solution of eq. (6), but it does reduce the loss in (6) at each step and seems to find good solutions. We repeat the alternation S times ($S = 25$ in our experiments). The complete pseudocode of our algorithm is given in Alg. 1.

4 Experimental evaluation

We evaluate our additive low-rank compression scheme and the corresponding learning algorithm using networks trained on two datasets: the CIFAR10 dataset containing 60K colored images of 28×28 in 10 classes, and the ImageNet-2012 dataset containing 1.2M various-sized colored images in 1 000 different classes. On CIFAR10 we train the batch-normalized version of the VGG16 network [16] that has 15.3M parameters, 313.73 MFLOPs, and achieving 6.46% test error. On ImageNet, we train a batch-normalized version of the AlexNet network [17] that has 62.3M weights, 1140 MFLOPs, and top-1/top-5 validation error of 42.29%/19.54%.

To compress both VGG16 and AlexNet using the number of FLOPs as our compression cost C, we run our algorithm using the augmented Lagrangian (which requires an additional step over the vector of Lagrange multipliers). Our additive low-rank scheme is applied to convolutional layers only (using combination of scheme 1 and 2, see Fig. 1), while fully-connected layers (in both networks, the few layers closest to the output) are compressed with a regular low-rank scheme. We run the algorithm for a range of λ values to explore the entire compression-error tradeoff curve. For VGG16, we run $T = 60$ LC iterations, where the L step was optimized

using SGD for 15 epochs with a learning rate of 0.0007 (decayed by 0.99 after each epoch) on batches of 128 images. For AlexNet, we run $T = 30$ LC iterations, where the L step was optimized using SGD for 10 epochs and with a learning rate of 0.001 (decayed by 0.9 after each epoch) on batches of 256 images. We used an exponential schedule for the μ values: at the tth LC iteration we used $\mu_t = a \times b^t$. For VGG16 we had $a = 2 \times 10^{-5}$ and $b = 1.2$; for AlexNet we had $a = 5 \times 10^{-4}$ and $b = 1.2$.

Results and comparison We report our compressed VGG16 and AlexNet models as tradeoff curves of FLOPs vs. test error in Fig. 2. We compare our additive low-rank compression to the methods where only low-rank schemes 1 or 2 were used [8]. We additionally report, as individual markers, recent results from the literature on reducing the number of FLOPs of VGG16 and AlexNet.

Our compressed models achieve considerable FLOPs reduction for the same accuracy levels compared to using only scheme 1 or 2, and also compared to other relevant compression methods. For instance, we compress the VGG16 network to have 52.52 MFLOPs ($\times$5.97 reduction) with no accuracy drop (6.40% test error); and we compress AlexNet to have 200.56 MFLOPs with top-1/top-5 error of 41.93/19.21% with a better tradeoff than schemes 1 and 2 (results of [8]).

5 Conclusion

We have proposed a new compression scheme for neural networks that additively combines low-rank decompositions using different matrix shapes. We formulated an optimization problem to jointly learn ranks and weights of these decompositions, and gave an effective algorithm based on the Learning-Compression framework. The algorithm alternates two steps: the L step of learning the neural network (a standard ML procedure) and the C step of compression of the neural network weights (formulated as a signal compression problem). We demonstrated that our additive combination improves the compression-error tradeoff across multiple neural networks, and outperforms other compression schemes. All scripts required to replicate our experiments are included in our LC algorithm toolbox [13], available at `https://github.com/UCMerced-ML/LC-model-compression`.

Acknowledgments We thank NVIDIA Corporation for several GPU donations.

References

[1] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. 43rd Int. Symposium on Computer Architecture (ISCA 2016)*.

[2] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[3] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 1943–1955, 2016.

[4] Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong, "Trained rank pruning for efficient deep neural networks," arXiv:1812.02402, Dec. 8 2018.

[5] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung, "Efficient neural network compression," in *Proc. of the 2019 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'19)*.

[6] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. of the 25th British Machine Vision Conference (BMVC 2014)*.

[7] Chong Li and C. J. Richard Shi, "Constrained optimization based low-rank approximation of deep neural networks," in *Proc. 15th European Conf. Computer Vision (ECCV'18)*.

[8] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "Low-rank compression of neural nets: Learning the rank of each layer," in *Proc. of the 2020 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'20)*.

[9] Miguel Á. Carreira-Perpiñán, "Model compression as constrained optimization, with application to neural nets. Part I: General framework," arXiv:1707.01209, July 5 2017.

[10] Miguel Á. Carreira-Perpiñán and Yerlan Idelbayev, "Model compression as constrained optimization, with application to neural nets. Part II: Quantization," arXiv:1707.04319, July 13 2017.

[11] Miguel Á. Carreira-Perpiñán and Yerlan Idelbayev, ""Learning-compression" algorithms for neural net pruning," in *Proc. of the 2018 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'18)*.

[12] Miguel Á. Carreira-Perpiñán and Arman Zharmagambetov, "Fast model compression," in *Bay Area Machine Learning Symposium (BayLearn 2018)*.

[13] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "A flexible, extensible software framework for model compression based on the LC algorithm," arXiv:2005.07786, May 15 2020.

[14] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "More general and effective model compression via an additive combination of compressions," Submitted.

[15] Jorge Nocedal and Stephen J. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering. Springer, second edition, 2006.

[16] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[18] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian, "Variational convolutional neural network pruning," in *Proc. of the 2019 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'19)*.

[19] Hao Li, Asim Kadav, Igor Durdanovic, and Hans P. Graf, "Pruning filters for efficient ConvNets," in *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*.

[20] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao, "HRank: Filter pruning using high-rank feature map," in *Proc. of the 2020 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'20)*.

[21] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann, "Towards optimal structured CNN pruning via generative adversarial learning," in *Proc. of the 2019 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'19)*.

[22] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. of the 2019 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'19)*.

[23] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. of the 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'16)*.

[24] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis, "NISP: Pruning networks using neuron importance score propagation," in *Proc. of the 2018 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'18)*.

[25] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan, "Approximated oracle filter pruning for destructive CNN width optimization," in *Proc. of the 36th Int. Conf. Machine Learning (ICML 2019)*.

[26] Jiashi Li, Qi Qi, Jingyu Wang, Ce Ge, Yujian Li, and Haifeng Sun, "OICSR: Out-in-channel sparsity regularization for compact deep neural networks," in *Proc. of the 2019 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'19)*.

[27] Zehao Huang and Naiyan Wang, "Data-driven sparse structure selection for deep neural networks," in *Proc. 15th European Conf. Computer Vision (ECCV'18)*.