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

Yerlan Idelbayev and **Miguel Á. Carreira-Perpiñán**
Dept. CSE, University of California, Merced
http://eecs.ucmerced.edu

The code is available at:
https://github.com/UCMerced-ML/LC-model-compression

Introduction: Low-rank for neural nets

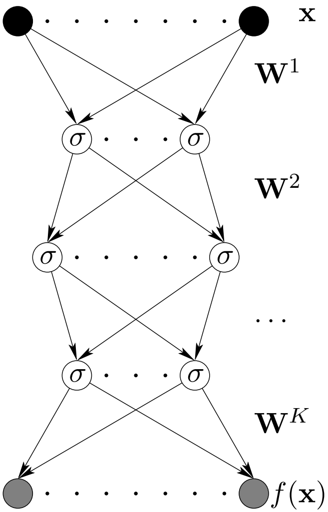


figure from Carreira-Perpiñán and Weiran Wang, arxiv:1212.5921

A K-layer neural net is a computational graph that computes output f for an input $\mathbf{x}$:

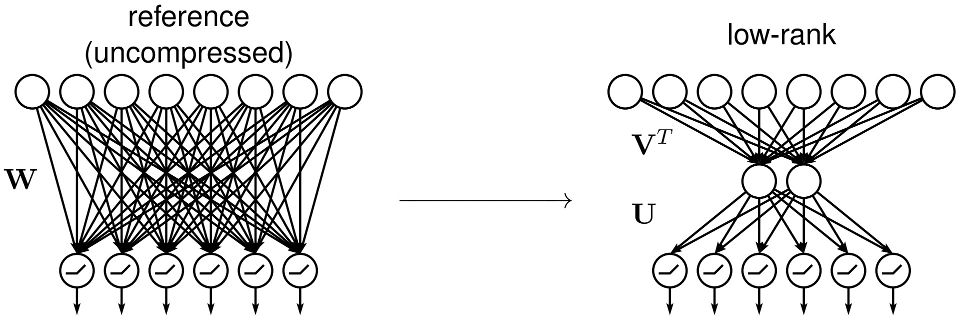
$$f(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{W}^K \ldots \sigma(\mathbf{W}^2 \sigma(\mathbf{W}^1 \mathbf{x})))$$

The weights $\mathbf{W} = \{\mathbf{W}^1, \ldots, \mathbf{W}^K\}$ are trained on a dataset of input-output pairs $(\mathbf{x}, \mathbf{y})$ to make the network output $f(\mathbf{x}; \mathbf{w})$ closer to the true output $\mathbf{y}$:

regression: $\min_{\mathbf{w}} \quad L(\mathbf{w}) = \sum_{\mathbf{x},\mathbf{y}} \|\mathbf{y} - f(\mathbf{x}; \mathbf{w})\|^2$

classification: $\min_{\mathbf{w}} \quad L(\mathbf{w}) = \sum_{\mathbf{x},\mathbf{y}} \text{CrossEntropy}(\mathbf{y}, f(\mathbf{x}; \mathbf{w}))$

Introduction: Low-rank for neural nets (cont.)



We replace a matrix $\mathbf{W}^i$ with some rank-r matrix

- Such matrix can be written as the product $\mathbf{U}\mathbf{V}^T$, i.e., $\mathbf{W} = \mathbf{U}\mathbf{V}^T$
  - For small values of r this reduces FLOPs and storage
  - Can achieve speed-up on any hardware (uses standard matrix-vector products)
- If ranks are known, training is not hard: simply decompose and then use SGD
- If ranks are not known, selection algorithms exists including many heuristics ones

Proposed scheme: Using additive combinations

Instead of single low-rank constraint we propose to use an additive combination

$$\mathbf{W} = \boldsymbol{\Theta}_1 + \boldsymbol{\Theta}_2 \tag{1}$$

where $\boldsymbol{\Theta}_1$ and $\boldsymbol{\Theta}_2$ are some low-rank matrices.

Unfortunately such a naive scheme does not enrich the compression.
Sum of low-rank matrcies can be always expressed a single low-rank matrix.

Proposed scheme: Using additive combinations and reshapes

To circumvent the limitation of the naive approach we introduce a reshaping function $\mathcal{R}$ which is similar to MATLAB's reshape function

Formally, assume we have a weight matrix $\boldsymbol{\Theta}$ *of shape* $m \times n$ *(same as* $\mathbf{W}$*). Let us define a reshape* $\mathcal{R}(\boldsymbol{\Theta})$ *to be a re-ordering of the* nm *elements of* $\boldsymbol{\Theta}$ *into some* $p \times q$ *sized matrix (with* $qp = mn$*), such that* $\mathcal{R}(\boldsymbol{\Theta})$ *contains the same set of elements as* $\boldsymbol{\Theta}$*, but in different order.*

For example, a reshape can be as follows:

$$\mathcal{R}\left(\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}\right) \longrightarrow \begin{bmatrix} 1 & 3 & 5 & 2 & 4 & 6 \\ 7 & 9 & 11 & 8 & 10 & 12 \end{bmatrix}$$

And we will impose low-rank on reshaped matrices!

Proposed scheme: Using additive combinations and reshapes (cont.)

Using reshaping functions, our final compression scheme is as follows:

$$\mathbf{W} = \mathbf{\Theta}_1 + \mathbf{\Theta}_2, \quad \operatorname{rank}\left(\mathcal{R}_1(\mathbf{\Theta}_1)\right) = r_1, \quad \operatorname{rank}\left(\mathcal{R}_2(\mathbf{\Theta}_2)\right) = r_2. \tag{2}$$

- Includes *regular* low-rank as a special case: set $\mathcal{R}_1$ to be identity reshape and set r_2 = 0.
- Forward pass ($\mathbf{W}\mathbf{x}$) can be efficiently parallelized: compute $\mathbf{\Theta}_1\mathbf{x}$ and $\mathbf{\Theta}_2\mathbf{x}$ independently in parallel and then sum.

How to train networks with the proposed additive low-rank scheme?

To efficiently compress a network, we need to use the best configuration of ranks over the layers. But there are combinatorial number of possible configurations!

To handle the combinatorial nature of the problem:

- we formulate a suitable optimization problem
- and give an efficient optimization algorithm based on Learning-Compression framework [1, 2, 3, 4, 5]

Problem formulation

Given a K-layer net with weights $\mathbf{W} = \{\mathbf{W}^1, \dots, \mathbf{W}^K\}$ trained on the loss L (e.g., cross-entropy), we formulate the following rank selection problem:

$$\begin{aligned} \min_{\mathbf{W},\boldsymbol{\Theta},\mathbf{r}} \quad & L(\mathbf{W}) + \lambda\, C(\boldsymbol{\Theta}, \mathbf{r}) \\ \text{s.t.} \quad & \mathbf{W}^k = \boldsymbol{\Theta}_1^k + \boldsymbol{\Theta}_2^k, \quad \text{for} \quad k = 1 \dots K, \\ & \operatorname{rank}\left(\mathcal{R}_1(\boldsymbol{\Theta}_1^k)\right) = r_1^k, \quad \operatorname{rank}\left(\mathcal{R}_2(\boldsymbol{\Theta}_2^k)\right) = r_2^k. \end{aligned} \tag{3}$$

Here, the term $\lambda\, C(\boldsymbol{\Theta}, \mathbf{r})$ controls the amount of compression and can target a specific cost of interest like FLOPs or storage.

Problem formulation: Compression cost function

The cost C is a function of the layers' ranks:

$$C(\mathbf{\Theta}, \mathbf{r}) = C(\mathbf{r}) = \sum_{k=1}^{K} (\alpha_1^k r_1^k + \alpha_2^k r_2^k) \tag{4}$$

for some constant values of α_1^k and α_2^k.

This definition is quite general. Consider the inference FLOPs of a fully connected layer with weights $\mathbf{W}$ of $m \times n$. The total FLOPs is:

$$\begin{aligned} \text{FLOPs}(\mathbf{W}\mathbf{x}) &= \text{FLOPs}(\mathbf{\Theta}_1\mathbf{x}) + \text{FLOPs}(\mathbf{\Theta}_2\mathbf{x}) \\ &= \underbrace{(m+n)}_{\text{const.}} \times r_1 + \underbrace{(m+n)}_{\text{const.}} \times r_2 \end{aligned}$$

Problem formulation

Given a K-layer net with weights $\mathbf{W} = \{\mathbf{W}^1, \dots, \mathbf{W}^K\}$ trained on the loss L (e.g., cross-entropy), we formulate the following rank selection problem:

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

This is a mixed-integer optimization problem. To solve it efficiently, we need an algorithm that can natively handle both integer and real-valued weights. To perform the optimization we use a learning-compression algorithm.

Optimization algorithm: Deriving the L and C steps

Let us bring the equality constraints into the objective using a penalty method and obtain an equivalent formulation (optimized as $\mu \to \infty$):

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

Under standard assumptions, the stationary point of (5) at $\mu \to \infty$ is the a feasible solution of the original problem (3).

Optimization algorithm: Deriving the L and C steps (cont.)

Let us now apply alternating optimization over variables $\mathbf{W}$ and $\{\boldsymbol{\Theta}, \mathbf{r}\}$:

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

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

Regular NN training independent of compression, typically solved by SGD

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

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

Actual compression step independent of NN weights and dataset.

Optimization algorithm: Solution of the C step

Due to the layerwise separability of the cost function C, the C-step problem separates over the layers into K smaller problems:

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

We are not aware of any closed-form solution for this problem. However, we can again use alternating optimization, over the groups of $\{\boldsymbol{\Theta}_1^k, r_1^k\}$ and $\{\boldsymbol{\Theta}_2^k, r_2^k\}$. In such case each subproblem does have a closed-form solution, involving SVD and selection over the ranks [4].

Optimization algorithm: Pseudocode

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

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

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

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

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

for $k = 1, \dots, K$ — C step

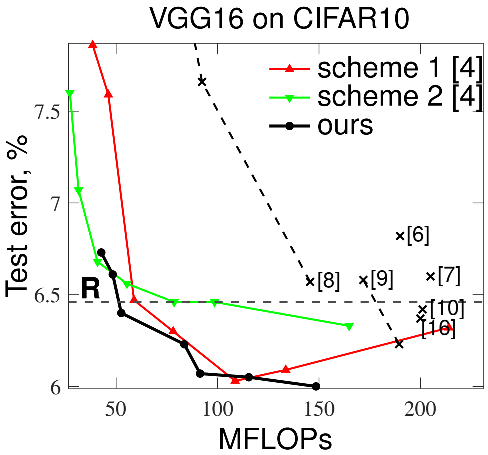
repeat for S times

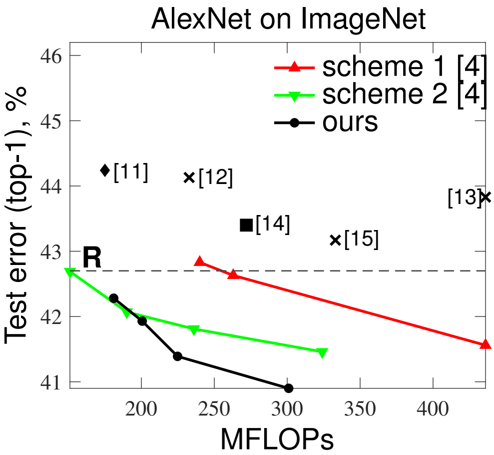
$$\boldsymbol{\Theta}_1^k, r_1^k \leftarrow \arg\min_{\boldsymbol{\Theta}_1^k, r_1^k} \lambda C_k(r_1^k, r_2^k) + \frac{\mu}{2} \left\|\mathbf{W}^k - \boldsymbol{\Theta}_2^k - \boldsymbol{\Theta}_1^k\right\|^2 \text{ s.t. } \operatorname{rank}\left(\mathcal{R}_1(\boldsymbol{\Theta}_1^k)\right) = r_1^k$$

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

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

Experiments





Code is available online

Our code is written in `Python` using `PyTorch`, and we make it available as part our extensible model compression framework (under `BSD 3-clause` license):

`https://github.com/UCMerced-ML/LC-model-compression`

Using the provided code, you will be able to:

- replicate all reported experiments
- compress your own models with our proposed scheme and many others.

But this library does much more than that. It is intended to support compression of an arbitrary model (not just neural nets) and an arbitrary compression technique. At the moment it offers the following:

- quantization (in various forms)
- pruning (in various forms)
- low-rank with automatic rank selection
- combinations of all the above

References

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

[2] 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.

[3] Miguel Á. Carreira-Perpiñán and Yerlan Idelbayev, ““Learning-compression” algorithms for neural net pruning,” in *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*, Salt Lake City, UT, June 18–22 2018, pp. 8532–8541.

[4] 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 Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20)*, Seattle, WA, June 14–19 2020, pp. 8046–8056.

[5] 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.

[6] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian, “Variational convolutional neural network pruning,” in *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, Long Beach, CA, June 16–20 2019, pp. 2780–2789.

[7] 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)*, Toulon, France, Apr. 24–26 2017.

[8] 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 Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20)*, Seattle, WA, June 14–19 2020, pp. 1526–1535.

[9] 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 Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, Long Beach, CA, June 16–20 2019, pp. 2790–2799.

[10] 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 Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, Long Beach, CA, June 16–20 2019, pp. 4340–4349.

[11] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng, “Quantized convolutional neural networks for mobile devices,” in *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*, Las Vegas, NV, June 26 – July 1 2016, pp. 4020–4028.

[12] 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 Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*, Salt Lake City, UT, June 18–22 2018, pp. 9194–9203.

[13] 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)*, Kamalika Chaudhuri and Ruslan Salakhutdinov, Eds., Long Beach, CA, June 9–15 2019, pp. 1607–1616.

[14] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung, “Efficient neural network compression,” in *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, Long Beach, CA, June 16–20 2019, pp. 12569–12577.

[15] Jiashi Li, Qi Qi, Jingyu Wang, Ce Ge, Yujian Li, Zhangzhang Yue, and Haifeng Sun, “OICSR: Out-in-channel sparsity regularization for compact deep neural networks,” in *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, Long Beach, CA, June 16–20 2019, pp. 7046–7055.

[16] Zehao Huang and Naiyan Wang, “Data-driven sparse structure selection for deep neural networks,” in *Proc. 15th European Conf. Computer Vision (ECCV'18)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, Eds., Munich, Germany, Sept. 8–14 2018, pp. 304–320.