Optimal Selection of Matrix Shape and Decomposition Scheme for Neural Network Compression

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

ICASSP2021
TORONTO
Canada
June 6-11, 2021
Metro Toronto Convention Centre

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

Introduction: Low-rank for neural nets

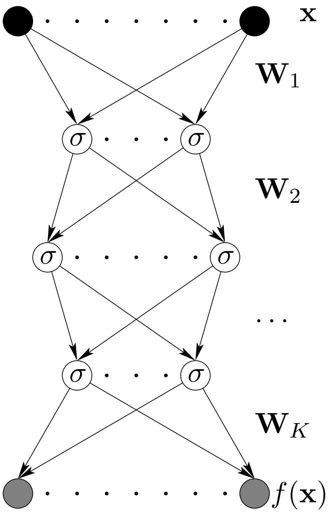


figure from Carreira-Perpiñán and Weiran Wang, arxiv:1212.5921

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

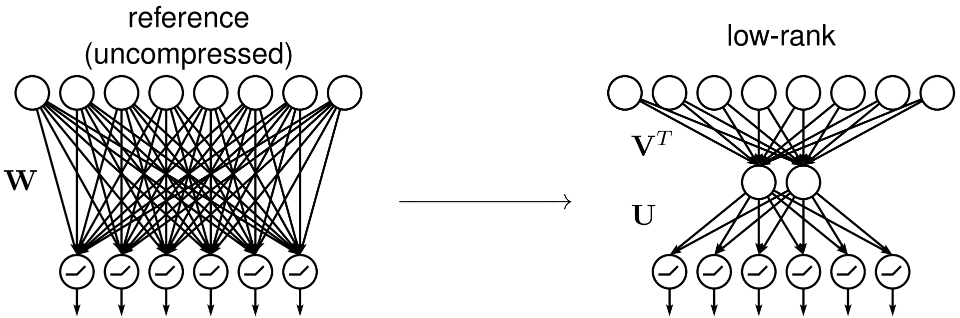
$$f(\mathbf{x};\mathbf{w}) = \sigma(\mathbf{W}_K \ldots \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x})))$$

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

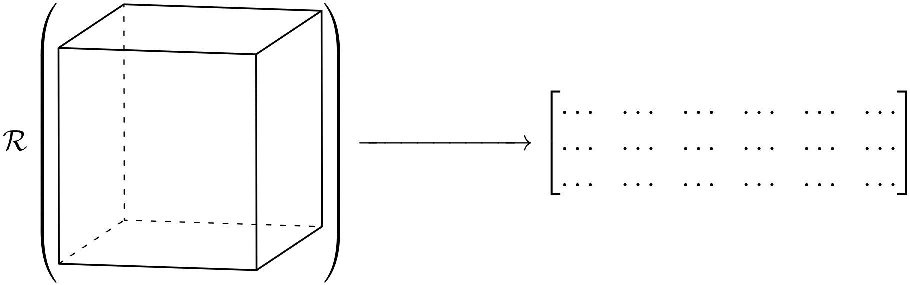
- Such matrix can be written as the product $\mathbf{U}\mathbf{V}^T$, i.e., $\mathbf{W} = \mathbf{U}\mathbf{V}^T$
  - For small values of r this reduces FLOPs and storage
  - Can achieve speed-up on any hardware (uses standard matrix-vector products)
- If ranks are known, training is not hard: simply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

What happens with non-matrix weights?

Weights are not necessarily come as matrices.

For example weights of convolutional layers are typically stored as NCHW or NHWC tensors.

To apply low-rank, we reshape the tensors into matrices!

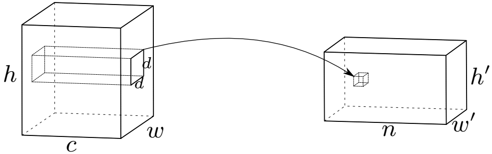


*This is known as matricization in tensor algebra.

More on reshapes: Efficient implementation

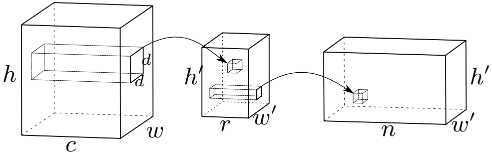
Some of the reshapes give a rise to efficient low-rank schemes.

Regular convolution



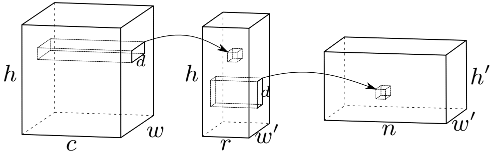
parameters: ncd^2 FLOPs: $ncd^2w'h'$

Low-rank using scheme 1 reshape



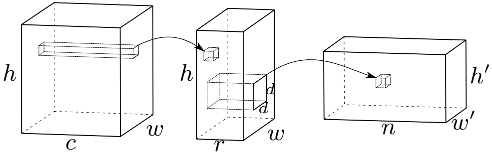
parameters: $r(cd^2 + n)$ FLOPs: $(cd^2 + n)rw'h'$

Low-rank using scheme 2 reshape



parameters: $r(cd + nd)$ FLOPs: $(ch + nh')rdw'$

Low-rank using scheme 3 reshape



parameters: $r(c + nd^2)$ FLOPs: $(cwh + nd^2w'h')r$

Which reshapes are the best? How to select them optimally?

- Historically a single fixed scheme was used throughout the NN for the compression
  - This is suboptimal!
- Can we select the best scheme per each layer?
- The problem involves selecting ranks as well.

Hard problem. There are combinatorial number of configurations of ranks and schemes. We tackle it by

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

Problem formulation

Given a K-layer net with weights $\mathbf{W} = \{\mathcal{W}_1, \dots, \mathcal{W}_K\}$ trained on the loss $\mathcal{L}$ (e.g., cross-entropy), we formulate the following rank and scheme selection problem:

$$\begin{aligned} \min_{\mathbf{W},\boldsymbol{\Theta},\mathbf{r},\mathbf{s}} \quad & \mathcal{L}(\mathbf{W}) + \lambda\, \mathcal{C}(\boldsymbol{\Theta}, \mathbf{r}) \\ \text{s.t.} \quad & \boldsymbol{\Theta}_k = \mathcal{R}(\mathcal{W}_k, s_k), \\ & \operatorname{rank}(\boldsymbol{\Theta}_k) = r_k, \quad \forall k = 1, \dots, K \end{aligned} \tag{1}$$

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

Problem formulation: Compression cost function

The cost $\mathcal{C}$ is a function of the layers' ranks:

$$\mathcal{C}(\boldsymbol{\Theta}, \mathbf{r}) = \mathcal{C}(\boldsymbol{\Theta}_1, r_1) + \cdots + \mathcal{C}(\boldsymbol{\Theta}_K, r_K).$$

- Can target storage and FLOPs of the model. Typically, cost per layer has form $\mathcal{C}(\boldsymbol{\Theta}_k, r_k) = \alpha \times r_k$ for some constant α.
- Can target the nuclear norm [6] of weight matrices instead: $\mathcal{C}(\boldsymbol{\Theta}_k, r_k) = \|\boldsymbol{\Theta}_k\|_*$.

Problem formulation

Given a K-layer net with weights $\mathbf{W} = \{\mathcal{W}_1, \dots, \mathcal{W}_K\}$ trained on the loss $\mathcal{L}$ (e.g., cross-entropy), we formulate the following rank and scheme selection problem:

$$\begin{aligned} \min_{\mathbf{W}, \boldsymbol{\Theta}, \mathbf{r}, \mathbf{s}} \quad & \mathcal{L}(\mathbf{W}) + \lambda \mathcal{C}(\boldsymbol{\Theta}, \mathbf{r}) \\ \text{s.t.} \quad & \boldsymbol{\Theta}_k = \mathcal{R}(\mathcal{W}_k, s_k), \\ & \operatorname{rank}(\boldsymbol{\Theta}_k) = r_k, \quad \forall k = 1, \dots, K \end{aligned}$$

This is a mixed-integer optimization problem over ranks and schemes.
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 apply a penalty method and obtain an equivalent formulation (with $\mu \to \infty$):

$$\min_{\mathbf{W},\boldsymbol{\Theta},\mathbf{r},\mathbf{s}} \quad \mathcal{L}(\mathbf{W}) + \lambda\, \mathcal{C}(\boldsymbol{\Theta}, \mathbf{r}) + \frac{\mu}{2} \sum_{k=1}^{K} \|\boldsymbol{\Theta}_k - \mathcal{R}(\mathcal{W}_k, s_k)\|_F^2 \tag{2}$$

$$\text{s.t.} \quad \operatorname{rank}(\boldsymbol{\Theta}_k) = r_k, \quad \forall k = 1, \dots, K.$$

Under standard assumptions, the stationary point of (2) at $\mu \to \infty$ is the stationary point (solution) of the original problem (1).

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

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

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

$$\min_{\mathbf{W}} \quad \mathcal{L}(\mathbf{W}) + \frac{\mu}{2} \sum_{k=1}^{K} \|\boldsymbol{\Theta}_k - \mathcal{R}(\mathcal{W}_k, s_k)\|_F^2.$$

  Regular NN training independent of compression, typically solved by SGD

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

$$\min_{\boldsymbol{\Theta}, \mathbf{r}, \mathbf{s}} \quad \lambda \mathcal{C}(\boldsymbol{\Theta}, \mathbf{r}) + \frac{\mu}{2} \sum_{k=1}^{K} \|\boldsymbol{\Theta}_k - \mathcal{R}(\mathcal{W}_k, s_k)\|_F^2$$

  Actual compression step independent of NN weights and dataset.

Optimization algorithm: Solution of the C step

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

$$\begin{aligned} \min_{\boldsymbol{\Theta}_k, r_k, s_k} \quad & \lambda \mathcal{C}(\boldsymbol{\Theta}_k, r_k) + \frac{\mu}{2} \|\boldsymbol{\Theta}_k - \mathcal{R}(\mathcal{W}_k, s_k)\|_F^2 \\ \text{s.t.} \quad & \operatorname{rank}(\boldsymbol{\Theta}_k) = r_k. \end{aligned} \tag{3}$$

Solution:

- For a fixed scheme s_k the solution is known in closed form for multiple costs $\mathcal{C}$
  - For storage and FLOPs, the solution involves SVD and enumeration [4]
  - For nuclear-norm cost, the solution involves singular value shrinkage [7]
- Therefore, to find global solution, we iterate over possible schemes and re-use steps for fixed scheme.

Optimization algorithm: Pseudocode

input K-layer neural net with weights $\mathbf{W} = \{\mathcal{W}_1, \dots, \mathcal{W}_K\}$,
hyperparameter λ, cost function $\mathcal{C}$, set of reshaping schemes $\{\mathcal{S}_1, \dots, \mathcal{S}_m\}$

$\mathbf{W} = (\mathcal{W}_1, \dots, \mathcal{W}_K) \leftarrow \arg\min_{\mathbf{W}} L(\mathbf{W})$ — reference net

$\mathbf{r} = (r_1, \dots, r_K) \leftarrow \mathbf{0}$ — ranks

$\mathbf{s} = (s_1, \dots, s_K) \leftarrow (\mathcal{S}_1, \dots, \mathcal{S}_1)$ — decomposition schemes

$\boldsymbol{\Theta} = (\boldsymbol{\Theta}_1, \dots, \boldsymbol{\Theta}_K) \leftarrow \mathbf{0}$ — reshaped weights

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

$$\mathbf{W} \leftarrow \arg\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}) + \frac{\mu}{2} \sum_{k=1}^{K} \|\boldsymbol{\Theta}_k - \mathcal{R}(\mathcal{W}_k, s_k)\|_F^2$$ — L step

for $k = 1, \dots, K$ — C step

for $s'_k = \mathcal{S}_1, \dots, \mathcal{S}_m$

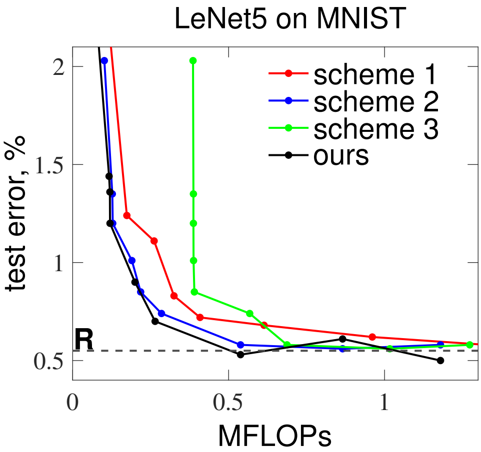
$$\boldsymbol{\Theta}'_k, r'_k \leftarrow \arg\min_{\boldsymbol{\Theta}_k, r_k} \lambda\, \mathcal{C}_k(r_k) + \frac{\mu}{2} \|\boldsymbol{\Theta}_k - \mathcal{R}(\mathcal{W}_k, s'_k)\|^2$$

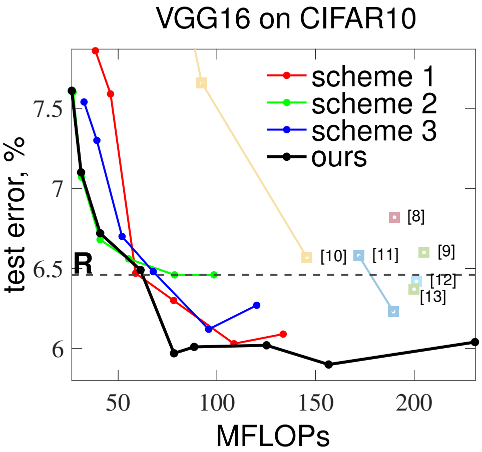
if $(\boldsymbol{\Theta}'_k, r'_k, s'_k)$ has a lower C-step objective **then**

$(\boldsymbol{\Theta}_k, r_k, s_k) \leftarrow (\boldsymbol{\Theta}'_k, r'_k, s'_k)$

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 and/or scheme 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] Maryam Fazel, *Matrix Rank Minimization with Applications*, Ph.D. thesis, Stanford University, Mar. 2002.

[7] Jian-Feng Cai, Emmanuel J. Candès, and Zuowei Shen, “A singular value thresholding algorithm for matrix completion,” *SIAM J. Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.