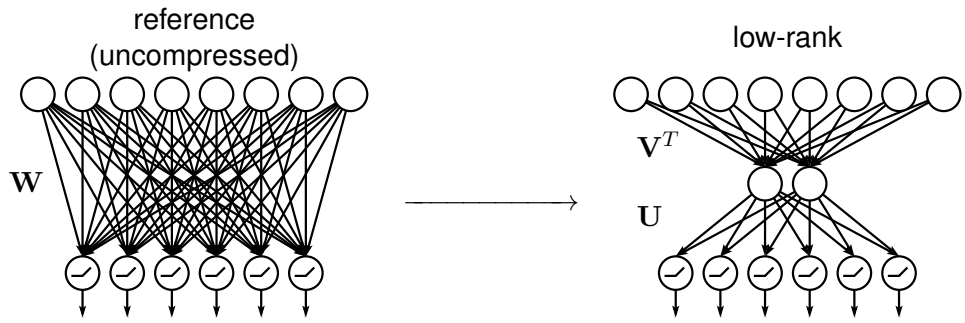
Beyond FLOPs in Low-rank Compression of Neural Networks: Optimizing Device-specific Inference Runtime

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

ICIP 2021

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

Introduction: Low-rank for neural nets



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

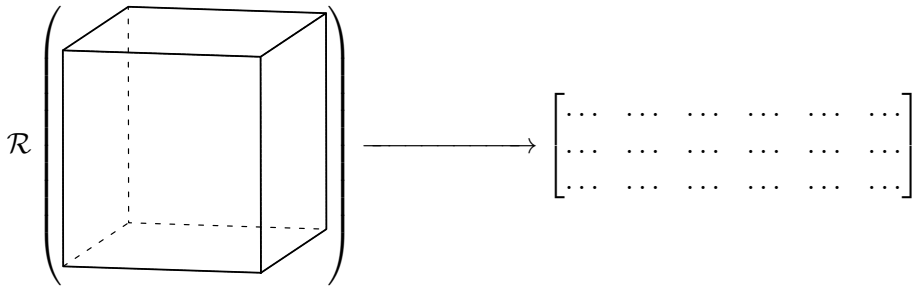
- Such matrix can be written as the product $\mathbf{U}\mathbf{V}^T$, i.e., $\mathbf{W} = \mathbf{U}\mathbf{V}^T$
  - For small values of r this reduces FLOPs and storage
  - Can achieve speed-up on any hardware (uses standard matrix-vector products)
- If ranks are known, training is not hard: simply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

What happens with non-matrix weights?

Weights do 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.

Can we apply low-rank to directly optimize inference time?

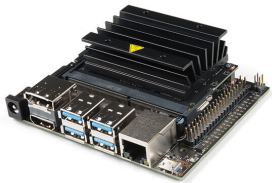
Historically, low-rank was used to reduce sizes and FLOPs of the models. But:

- fewer FLOPs not necessarily mean faster runtime!
- Can we select the ranks per each layer to minimize on-device runtime? (requires on-device measurements)

Hard problem There are combinatorial number of ranks and corresponding on-device measurements. We tackle it by

- building an accurate and fast to compute runtime model
- formulating a suitable optimization problem
- and giving an efficient optimization algorithm based on Learning-Compression framework [1, 2, 3, 4, 5, 6, 7, 8, 9]

Our target device :



Jetson Nano

CPU	4-core ARM Cortex-A57, 1.4 GHz
GPU	128 CUDA cores at 0.9 GHz
RAM	4 GB 64-bit LPDDR4, 1.6 GHz
OS	Ubuntu 18.04.5 LTS
Kernel	GNU/Linux 4.9.140-tegra
Storage	128 GB microSDXC memory card

Part I: Device runtime model

Let's define the runtime $\mathcal{R}(\mathbf{W})$ as the time to process a single image through a K-layer net with weights $\mathbf{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_K\}$.

- runtime is function of layer's ranks
- runtime can be directly measured on device
- assuming R ranks per layer, there are R^K different configurations to measure

We model the runtime as the sum of inferences through individual layers:

$$\mathcal{R}(\mathbf{W}) = \mathcal{R}(\mathbf{r}) = \mathcal{R}_1(r_1) + \mathcal{R}_2(r_2) + \cdots + \mathcal{R}_K(r_K). \tag{1}$$

In reality $\mathcal{R}(\mathbf{W}) \leq$ RHS: when computational graph is executed optimally, some weights and inputs can be prefetched and layer-to-layer computations can be pipelined

This model allows to obtain runtime estimate $\mathcal{R}(\mathbf{W})$ much more efficiently:

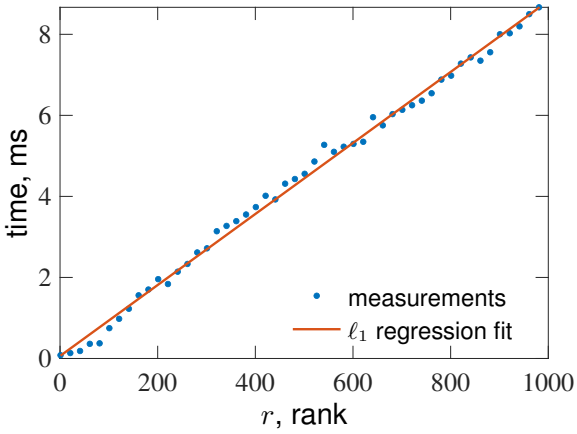
- only need to measure R different rank configurations for each of the K layers
- total required measurements: $R \times K$ (vs R^K)

Part I: Device runtime model (cont.)

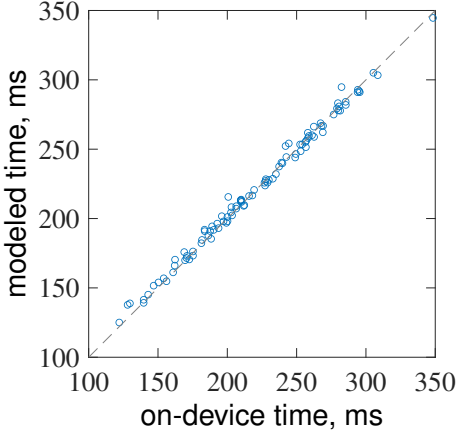
Even RK on-device measurements are time consuming and noisy, thus:

- for each layer we run measurements for equally spaced set of ranks (e.g., $r = 1, 10, 20, \dots$)
- fit ℓ_1 regression on the measurements to interpolate and reduce noise

$\mathcal{R}_7(r)$ measurements for AlexNet



True vs. modeled inference time



Part II: Problem formulation

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

$$\begin{aligned} \min_{\mathbf{W},\mathbf{r}} \quad & \mathcal{L}(\mathbf{W}) + \lambda \mathcal{R}(\mathbf{r}) \\ \text{s.t.} \quad & \operatorname{rank}(\mathbf{W}_k) = r_k, \quad k = 1, \ldots, K. \end{aligned} \tag{2}$$

Here, the term $\lambda\,\mathcal{R}(\mathbf{r})$ controls the tradeoff between on-device inference speed and model loss.

Part II: Optimization algorithm

Let us introduce auxiliary variable $\mathbf{\Theta}_k$ for each $\mathbf{W}_k$ as:

$$\operatorname{rank}(\mathbf{W}_k) = r_k \iff \mathbf{W}_k = \mathbf{\Theta}_k,\ \operatorname{rank}(\mathbf{\Theta}_k) = r_k,$$

And apply a penalty method and obtain an equivalent formulation (with $\mu \to \infty$):

$$\begin{aligned} \min_{\mathbf{W},\mathbf{\Theta},\mathbf{r}} \quad & \mathcal{L}(\mathbf{W}) + \frac{\mu}{2} \sum_{k=1}^{K} \|\mathbf{W}_k - \mathbf{\Theta}_k\|^2 + \lambda \mathcal{R}(\mathbf{r}) \\ \text{s.t.} \quad & \operatorname{rank}(\mathbf{\Theta}_k) = r_k, \quad k = 1, \ldots, K. \end{aligned} \tag{3}$$

Here, we use Quadratic Penalty for the ease of presentation, however, in practice we use augmented Lagrangian with the additional step over Lagrange multipliers

Part II: Optimization algorithm (deriving the L and C steps)

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:

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

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:

$$\begin{aligned} \min_{\boldsymbol{\Theta}, \mathbf{r}} \quad & \frac{\mu}{2} \sum_{k=1}^{K} \|\mathbf{W}_k - \boldsymbol{\Theta}_k\|^2 + \lambda \mathcal{R}(\mathbf{r}) \\ \text{s.t.} \quad & \operatorname{rank}(\boldsymbol{\Theta}_k) = r_k, \quad k = 1, \ldots, K \end{aligned}$$

Actual compression step independent of NN weights and dataset.

Part II: Optimization algorithm (solution of the C step)

Due to the layerwise separability of the runtime function $\mathcal{R}(\mathbf{r})$, the C-step problem separates over the layers into K smaller problems:

$$\begin{aligned} \min_{\boldsymbol{\Theta}_k, r_k} \quad & \frac{\mu}{2}\|\mathbf{W}_k - \boldsymbol{\Theta}_k\|^2 + \lambda \mathcal{R}_k(r_k) \\ \text{s.t.} \quad & \operatorname{rank}(\boldsymbol{\Theta}_k) = r_k. \end{aligned} \tag{4}$$

Solution:

- Solution of this problem requires an SVD and enumeration over the ranks. More details are in the main paper.

Part II: Optimization algorithm (pseudocode)

input K-layer neural net with weights $\mathbf{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_K\}$, hyperparameter λ, device runtime model $\mathcal{R}$.

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

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

$\mathbf{\Theta} = (\mathbf{\Theta}_1, \ldots, \mathbf{\Theta}_K) \leftarrow \mathbf{0}$ — auxiliary variables

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

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

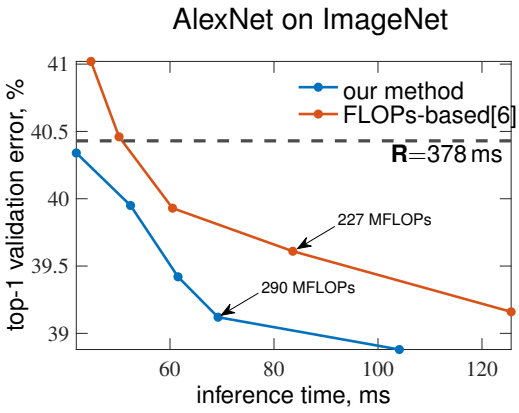
for $k = 1, \ldots, K$ — C step

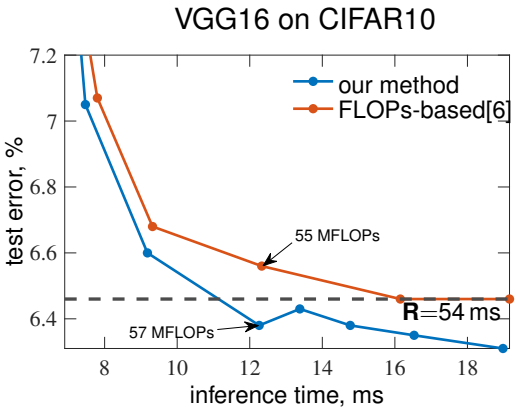
$$\mathbf{\Theta}_k, r_k \leftarrow \arg\min_{\mathbf{\Theta}_k, r_k} \frac{\mu}{2} \|\mathbf{\Theta}_k - \mathbf{W}_k\|^2 + \lambda \mathcal{R}_k(r_k)$$

if $\|\mathbf{W} - \mathbf{\Theta}\|$ is small enough **then** exit the loop

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

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] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "Neural network compression via additive combination of reshaped, low-rank matrices," in *Proc. Data Compression Conference (DCC 2021)*, Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagrista, and James A. Storer, Eds., Online, Mar. 23–26 2021, pp. 243–252.

[2] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "Optimal selection of matrix shape and decomposition scheme for neural network compression," in *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'21)*, Toronto, Canada, June 6–11 2021, pp. 3250–3254.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "An empirical comparison of quantization, pruning and low-rank neural network compression using the LC toolkit," in *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, July 18–22 2021.

[9] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "More general and effective model compression via an additive combination of compressions," in *Proc. of the 32nd European Conf. Machine Learning (ECML–21)*, Bilbao, Spain, Sept. 13–17 2021.