Low Rank Compression of Neural Networks: LC Algorithms and Open-source Implementation

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

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

Machine learning and neural networks

Most of this talk applies to ML models in general, however, we focus on neural networks

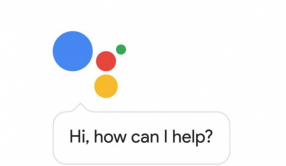
Neural networks have established state-of-the-art performance nearly in every machine learning task:

- Natural Language Processing (NLP): dialog systems, translators...
- Computer vision: image and video recognition, classification...
- Speech processing: speech-to-text, audio synthesis...
- Various signal enhancement tasks (photo, audio, video)

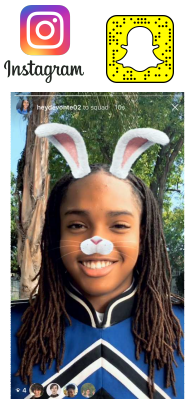
The models trained on these tasks have significant practical importance

Some famous neural network use cases

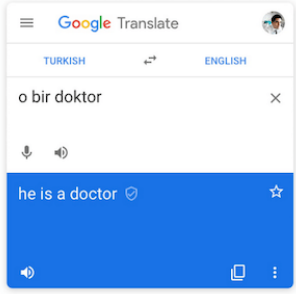
Voice Assistants



Photo/Video/AR



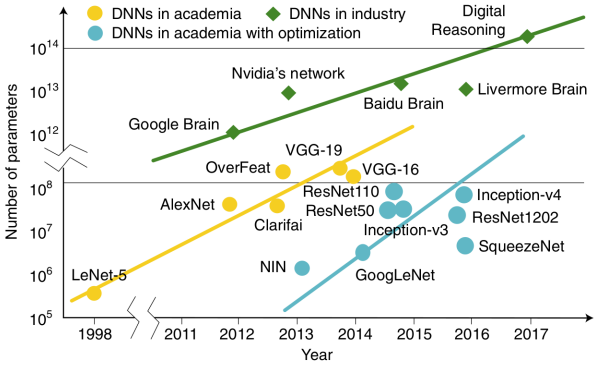
Translators



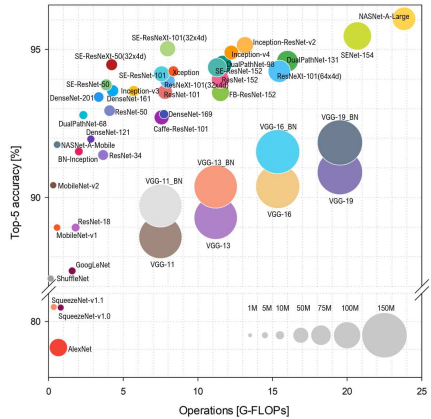
The images are obtained from official websites or blogposts of the services.

Current mobile devices contain dozens of neural networks, some of them running non-stop!

The improvement of NN performance



Number of parameters in NN over the years



FLOPs vs Top-5 accuracy on ImageNet

Why? The improvements are attributed to several factors:

- more data and more compute
- better software with frameworks like `PyTorch`, `TensorFlow`, `MxNet`
- better algorithms and vast amount of collected empirical knowledge

The top left figure is obtained from Xu et al. (doi: 10.1038/s41928-018-0059-3), the top right figure is obtained from Bianco et al. (doi: 10.1109/ACCESS.2018.2877890)

The deployment of NN

Two practical regimes of deployment

Cloud deployment:

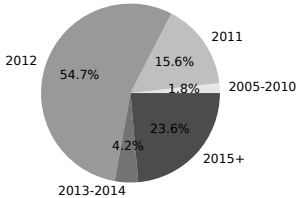
- access to powerful hardware
- the hardware can be chosen
- high-end hardware is typically allocated for training only
- running at scale is expensive

Edge and mobile deployment:

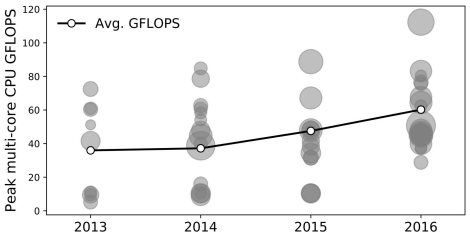
- hardware can not be chosen
- stringent constraints:
  - processing capabilities (CPU, GPU)
  - power consumption and battery life
  - SLA for responsiveness (for usability)
  - for mobile devices: most hardware is old and low-end

A study from Facebook over its mobile users

Most of the mobile devices in use are old, in 2018 only 25% of users had CPUs designed later than 2013.



Most of the mobile devices in use are low-end:



The figures are obtained from Wu et al. (doi: 10.1109/HPCA.2019.00048)

Model compression

There is a mismatch between requirements of current deep models and capabilities of end-user hardware.

Research question:

- How to obtain a smaller (compressed) neural network that has as close performance (e.g., accuracy) as possible to the original large model?

We will formulate the compression problem as a constrained optimization and give an efficient algorithm based on solid optimization principles (the Learning-Compression algorithm)

In this talk we show the application of LC for the problems of low-rank compression of neural networks.

Model compression: detour on neural networks

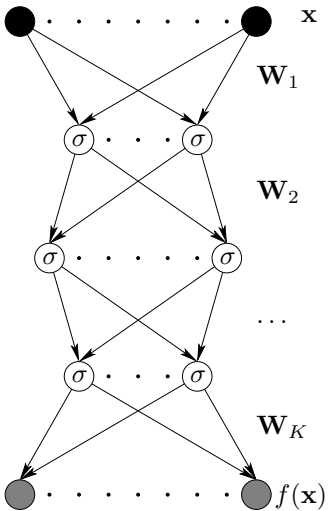


figure from Carreira-Perpiñán and Weiran Wang, arxiv:1212.5921

A neural network with K layers is a computational graph comprised of K linear and non-linear transforms applied to input $\mathbf{x}$:

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

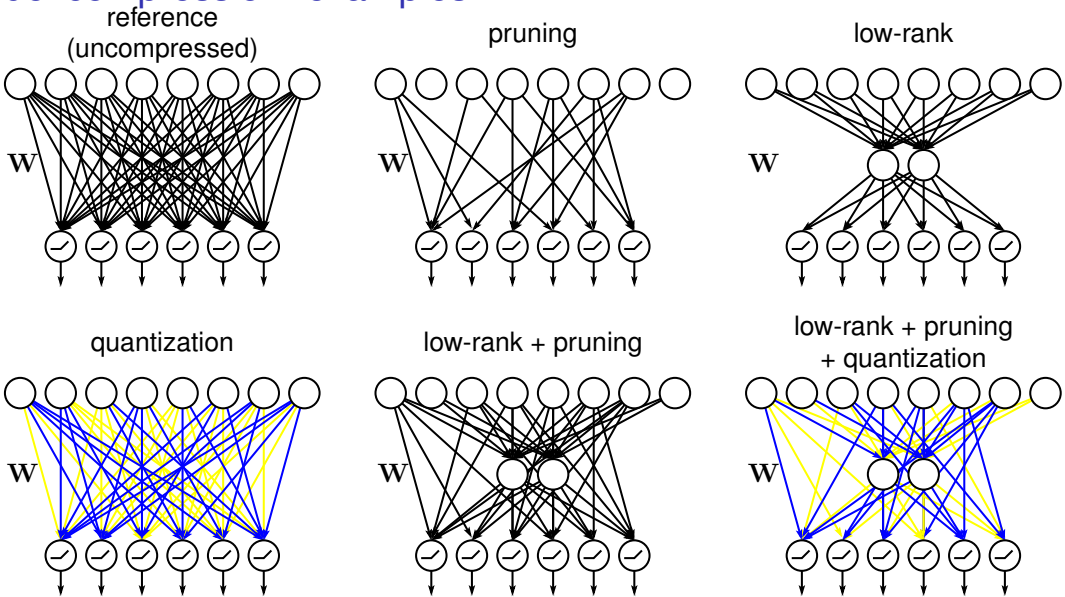
The weights $\mathbf{w} = \{\mathbf{W}_1, \dots, \mathbf{W}_K\}$ are trained on a dataset of input-output pairs $(\mathbf{x}, \mathbf{y})$ to make the network output $f(\mathbf{x};\mathbf{w})$ closer to the true output $\mathbf{y}$:

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

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

Typically, the networks are optimized using the stochastic gradient descent (SGD): a procedure that uses an estimate of the true gradient ($\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w})$) computed on training mini-batches.

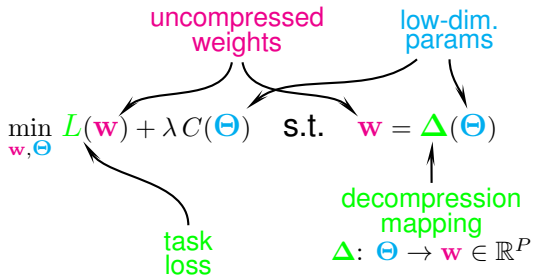
Model compression: examples



Images are from the slides of Miguel Á. Carreira-Perpiñán

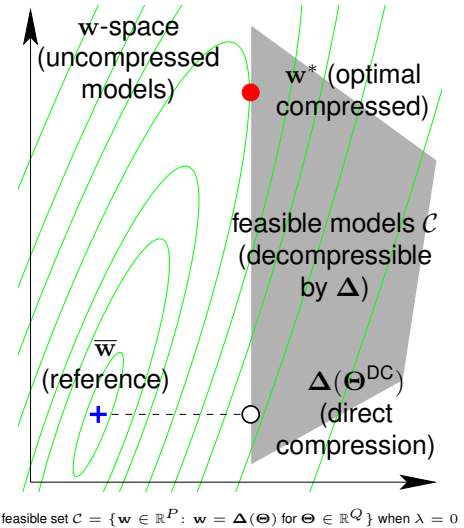
The LC algorithm: general formulation

Given a network with weights $\mathbf{w}$ and loss L:



Compression details are abstracted in $\boldsymbol{\Delta}(\boldsymbol{\Theta})$:

- low-rank: $\boldsymbol{\Delta}(\boldsymbol{\Theta}) = \mathbf{U}\mathbf{V}^T$ where $\boldsymbol{\Theta} = \{\mathbf{U}, \mathbf{V}\}$
- pruning: $\boldsymbol{\Delta}(\boldsymbol{\Theta}) = \boldsymbol{\Theta}$ s.t. $\|\boldsymbol{\Theta}\|_0 \leq \kappa$



The LC algorithm (cont.)

Reformulate using penalty methods and optimize the following while driving $\mu \to \infty$:

$$\min_{\mathbf{w},\boldsymbol{\Theta}} \quad L(\mathbf{w}) + \lambda\, C(\boldsymbol{\Theta}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2$$

Apply alternating optimization wrt $\mathbf{w}$ and $\boldsymbol{\Theta}$, which gives the (LC) algorithm:

- Learning (L) step:

$$\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2$$

  - This is a regular training of the model, but with a quadratic regularization term.
  - L step is independent of compression mechanism.
  - We will use SGD and standard NN software
- Compression (C) step:

$$\min_{\boldsymbol{\Theta}} \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2 + \lambda\, C(\boldsymbol{\Theta})$$

  - For $\lambda = 0$ the C step has a form of optimal projection of the weights
  - Does not involve the dataset (no L term).
  - Many well studied cases with fast solutions

The LC algorithm: pseudocode

input training data and model with parameters $\mathbf{w}$

$\mathbf{w} \leftarrow \overline{\mathbf{w}} = \arg\min_{\mathbf{w}} L(\mathbf{w})$ — pretrained model

$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}^{\mathrm{DC}} = \boldsymbol{\Pi}(\overline{\mathbf{w}})$ — init compression

$\boldsymbol{\beta} \leftarrow \mathbf{0}$

for $\mu = \mu_0 < \mu_1 < \cdots < \infty$

  $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta}) - \frac{1}{\mu}\boldsymbol{\beta}\|^2$ — L step

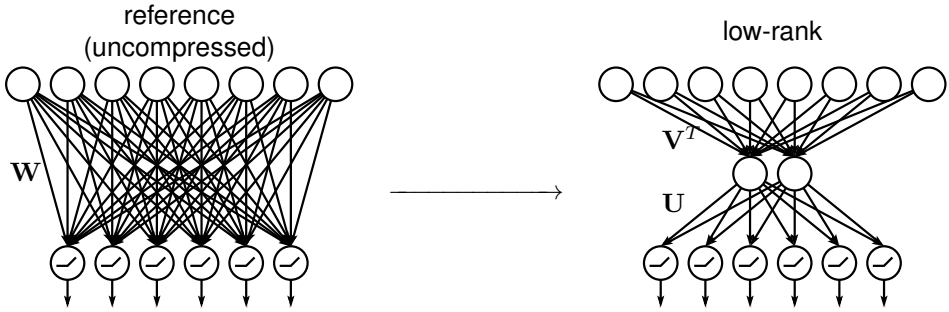
  $\boldsymbol{\Theta} \leftarrow \arg\min_{\boldsymbol{\Theta}} \frac{\mu}{2}\|\mathbf{w} - \frac{1}{\mu}\boldsymbol{\beta} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2 + \lambda C(\boldsymbol{\Theta})$ — C step

  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \mu(\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta}))$ — multipliers step

  if $\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|$ is small enough **then** exit the loop

return $\mathbf{w}$, $\boldsymbol{\Theta}$

Low-rank compression with rank selection



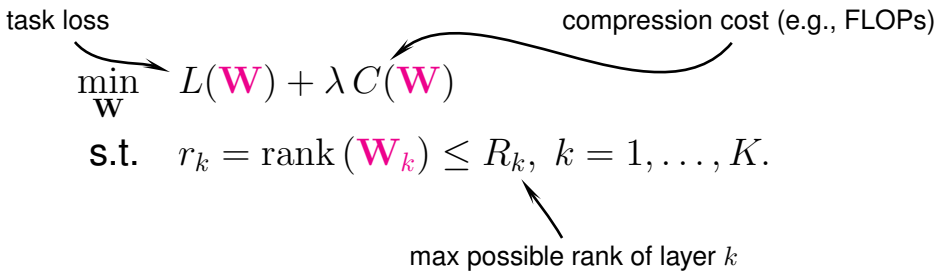
We replace a matrix $\mathbf{W}$ with rank-r matrix

- Such a low-rank matrix can be written as the product $\mathbf{U}\mathbf{V}^T$
  - For small values of r this reduces FLOPs and storage
  - Low-rank decomposition can achieve speed-up on any hardware (unlike weight pruning)
- If ranks are known, training the low-rank models is not hard: use SGD
- Selecting the right rank for each layer is challenging
  for K-layer network, with R ranks per layer there are R^K combinations to try

We use the LC algorithm to jointly learn both weights and ranks.

Low-rank compression: formulation of the problem

For a K-layer deep net with weights $\mathbf{W} = (\mathbf{W}_1, \dots, \mathbf{W}_K)$. Our compression problem is defined as:



- $\lambda C(\mathbf{W})$ controls the tradeoff between model performance and compression
- mixed-integer optimization
- the formulation does not allow to immediately use the LC algorithm

Low-rank compression: cost function

The compression cost C measures the model size/FLOPs and is defined as

$$C(\mathbf{W}) = C(\mathbf{r} = \{r_1, \ldots, r_K\}) = C_1(r_1) + \cdots + C_K(r_K).$$

Here, $C_k(r_k)$ is cost of compressing layer k using rank r_k. Assume, $\mathbf{W}_k$ is $m \times n$ matrix:

- size: $C_k(r_k) = m \times r_k + r_k \times n$
- FLOPs: $C_k(r_k) = (m \times r_k + r_k \times n) \times M$ where M is #times filter (layer) has been applied

Low-rank compression: optimization

We introduce auxiliary variables $\boldsymbol{\Theta}_k$ with a constraint of $\mathbf{W}_k = \boldsymbol{\Theta}_k$ for $k = 1, \ldots, K$:

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

and then apply penalty to equality constraints (with $\mu \to \infty$)

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

Low-rank compression: optimization (cont.)

After our simple manipulations, we obtained the following:

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

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

Now, the alternation over $\mathbf{W}$ and $\{\boldsymbol{\Theta}, \mathbf{r}\}$ gives us the LC algorithm:

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

same old L step, will be handled by SGD

C step: $\min_{\boldsymbol{\Theta},\mathbf{r}} \frac{\mu}{2}\sum_{k=1}^{K} \|\mathbf{W}_k - \boldsymbol{\Theta}_k\|^2 + \lambda C(\mathbf{r})$ s.t. $r_k \leq R_k \quad \forall k = 1, \ldots, K$

C step separates over layers into smaller subproblems

Low-rank compression: C step

$$\min_{\boldsymbol{\Theta},\mathbf{r}} \frac{\mu}{2} \sum_{k=1}^{K} \|\mathbf{W}_k - \boldsymbol{\Theta}_k\|^2 + \lambda \sum_{k=1}^{K} C(r_k), \quad \text{s.t.} \quad r_k \leq R_k, \quad k = 1, \ldots, K$$

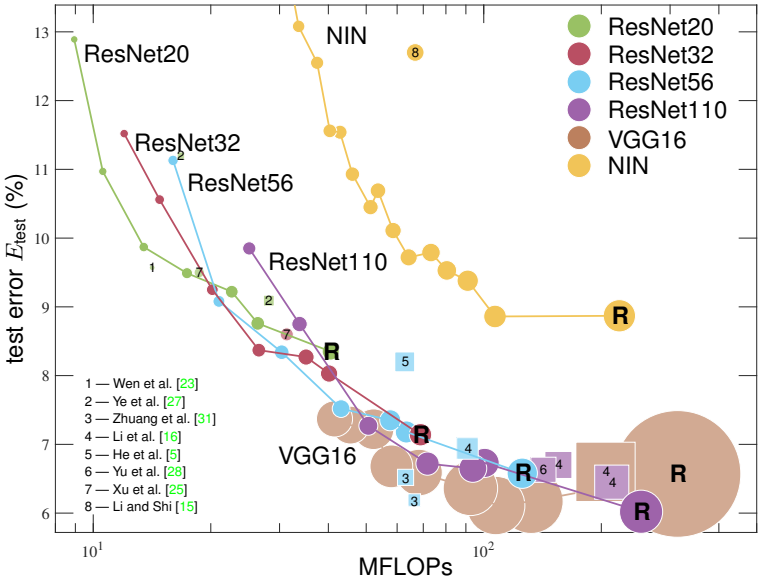
The C step problem separates over layers $k = 1, \ldots, K$ into:

$$\min_{\boldsymbol{\Theta}_k, r_k} \frac{\mu}{2} \|\mathbf{W}_k - \boldsymbol{\Theta}_k\|^2 + \lambda C(r_k) \quad \text{s.t.} \quad r_k = \operatorname{rank}(\boldsymbol{\Theta}_k) \leq R_k$$

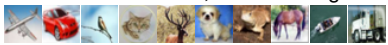
Solution: If the rank of decomposition (r_k) is fixed, this reduces to best r_k-rank approximation problem of a matrix. The solution of "best rank" approximation is given by Eckhart-Young theorem and computed using SVD. However, we have several ranks to check: $r_k = 1 \ldots R_k$: we simply enumerate over every possible rank and chose the one corresponding to lowest loss. No need to compute multiple SVDs, one is sufficient.

Behavior: We automatically select the best rank for our overall compression cost.

Low-rank compression: experiments on CIFAR10 networks



CIFAR10: 10 classes, 32×32 images:



50K in train set, 10K in test set

Error-compression space of test error, inference FLOPs and number of parameters (ball size for each net). Different networks have different colors. **R** — reference network.

Results of our algorithm over different λ values for a given network span a curve, shown as connected circles

Other published results using low-rank are shown as isolated circles; filter pruning results shown as isolated squares.

Low-rank: AlexNet on ImageNet experiments

The ImageNet task asks to classify a colored image into one of the 1000 classes.
AlexNet is a convolutional network with 60M parameters

	MFLOPs	top-1	top-5	ρ_{FLOPs}
Caffe-AlexNet [11]	724	42.70	19.80	1.00
Kim et al. [13], Tucker	272	n/a	21.67	2.66
Tai et al. [21], scheme 2	185	n/a	20.34	3.90
Wen et al. [23], scheme 1	269	n/a	20.14	2.69
Kim et al. [12], scheme 2	272	43.40	20.10	2.66
Yu et al. [28], filter prun.	232	44.13	n/a	3.12
Li et al. [17], filter prun.	334	43.17	n/a	2.16
Ding et al. [3], filter prun.	492	43.83	20.47	1.47
our scheme 1, $\lambda = 0.20$	231	41.56	18.72	3.13
our scheme 2, $\lambda = 0.20$	**152**	**41.03**	**18.23**	**4.78**

Compression with our algorithm vs published work using low-rank methods and structured pruning.
ρ_{FLOPs} — reduction in FLOPs.

Can we apply low-rank to optimize inference time?

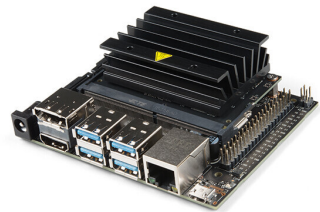
Historically, low-rank was used to reduce sizes and FLOPs of the models. But:

- 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

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

Inference-targeted low-rank: 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

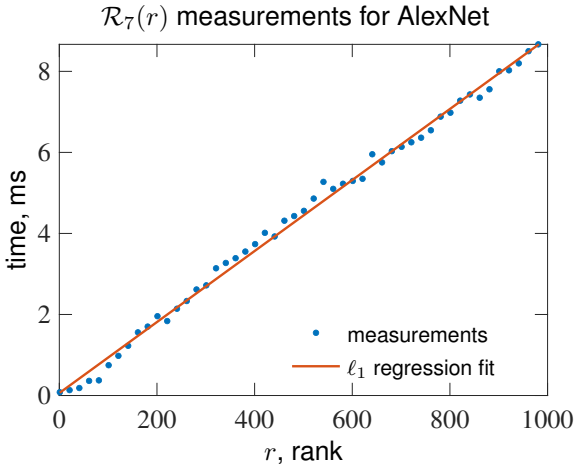
This model allows to obtain runtime estimate $\mathcal{R}(\mathbf{W})$ much more efficiently:

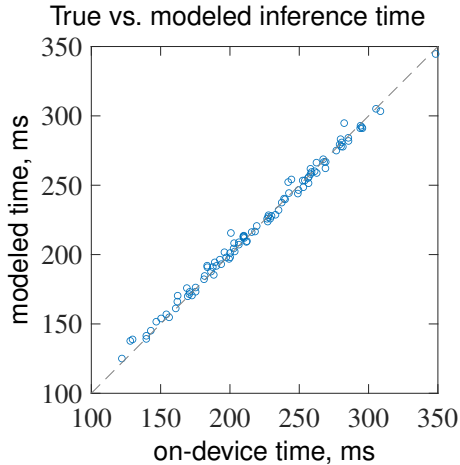
- only need to measure R different rank configurations for each of the K layers
- total required measurements: $R \times K$ (vs R^K)

Inference-targeted low-rank: 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., $r = 1, 10, 20, \ldots$)
- fit ℓ_1 regression on the measurements to interpolate and reduce noise





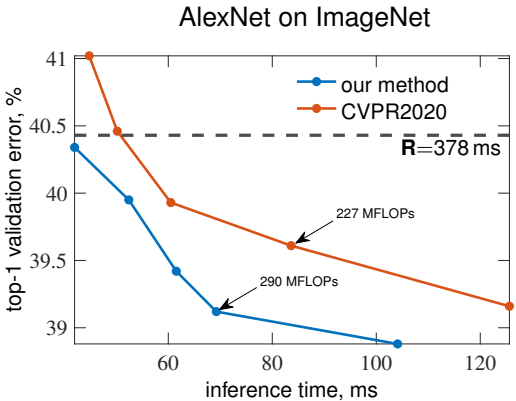
Inference-targeted low-rank: Problem formulation

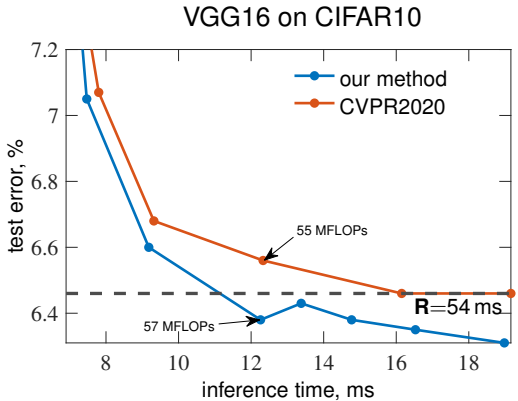
Given a K-layer net with weights $\mathbf{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_K\}$ trained on the loss L (e.g., cross-entropy), we formulate the following device-dependent rank selection problem:

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

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

Inference-targeted low-rank: Experiments





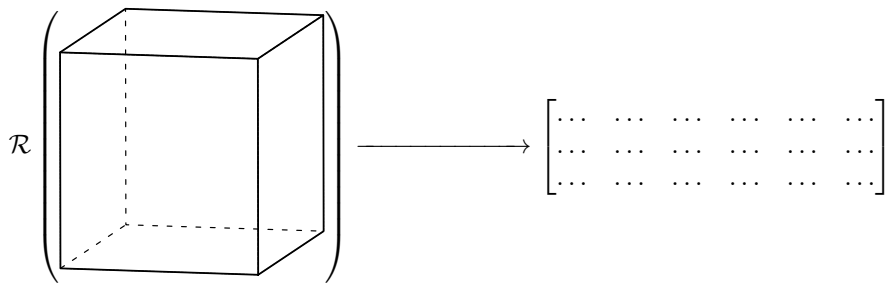
- optimizing for latency beats optimizing for FLOPs in terms of final latency (duh?)

Low-rank: 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!

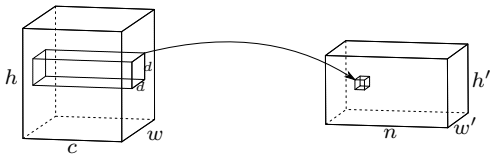


*This is known as matricization in tensor algebra.

More on reshapes: Efficient implementation

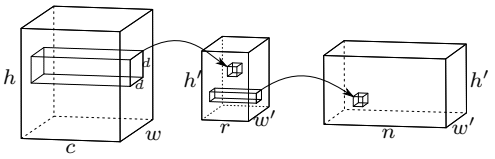
Some of the reshapes give a rise to efficient low-rank schemes.

Regular convolution



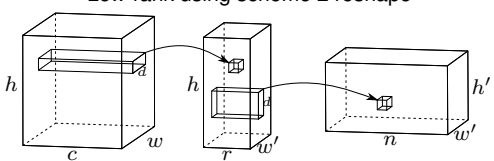
parameters: ncd^2 FLOPs: $ncd^2w'h'$

Low-rank using scheme 1 reshape



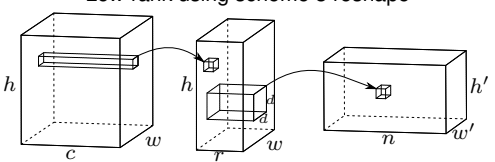
parameters: $r(cd^2 + n)$ FLOPs: $(cd^2 + n)rw'h'$

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
  - In our CVPR2020 work, we used fixed scheme throughout.
  - This is suboptimal!
- Can we select the best scheme per each layer?
- The problem involves selecting ranks as well.

Hard problem. There are a combinatorial number of configurations of ranks and schemes.

However, learning-compression algorithm can solve it too.

Low-rank with scheme selection: Problem formulation

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

$$\begin{aligned} \min_{\mathbf{W},\boldsymbol{\Theta},\mathbf{r},\mathbf{s}} \quad & L(\mathbf{W}) + \lambda\, C(\boldsymbol{\Theta}, \mathbf{r}) \\ \text{s.t.} \quad & \boldsymbol{\Theta}_k = \mathcal{R}(\mathcal{W}_k, s_k), \\ & \operatorname{rank}(\boldsymbol{\Theta}_k) = r_k, \quad \forall k = 1, \ldots, K \end{aligned} \tag{3}$$

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.

Low-rank with scheme selection: Deriving the L and C steps

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

$$\begin{aligned} \min_{\mathbf{W},\boldsymbol{\Theta},\mathbf{r},\mathbf{s}} \quad & L(\mathbf{W}) + \lambda\, C(\boldsymbol{\Theta},\mathbf{r}) + \frac{\mu}{2}\sum_{k=1}^{K} \|\boldsymbol{\Theta}_k - \mathcal{R}(\mathcal{W}_k, s_k)\|_F^2 \\ \text{s.t.} \quad & \operatorname{rank}(\boldsymbol{\Theta}_k) = r_k, \quad \forall k = 1,\dots,K. \end{aligned} \tag{4}$$

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:

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

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

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

Low-rank with scheme selection: Solution of the C step

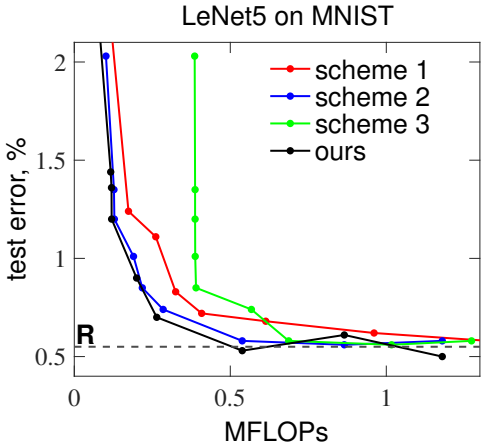
Due to the layerwise separability of cost function, the C-step problem separates over the layers into K smaller problems:

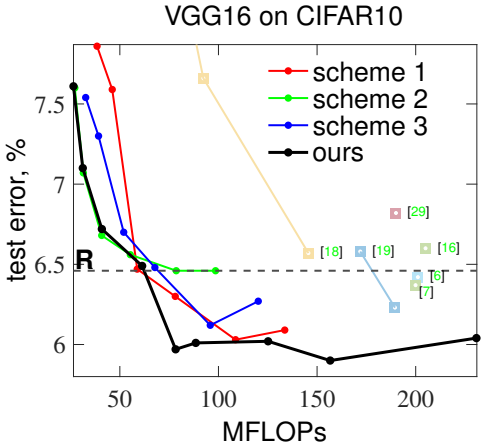
$$\begin{aligned} \min_{\boldsymbol{\Theta}_k, r_k, s_k} \quad & \lambda\, C(\boldsymbol{\Theta}_k, r_k) + \frac{\mu}{2}\|\boldsymbol{\Theta}_k - \mathcal{R}(\mathcal{W}_k, s_k)\|_F^2 \\ \text{s.t.} \quad & \operatorname{rank}(\boldsymbol{\Theta}_k) = r_k. \end{aligned} \tag{5}$$

Solution:

- For a fixed scheme s_k the solution is known in closed form for multiple costs C
  - For storage and FLOPs, the solution involves SVD and enumeration [8]
  - For nuclear-norm cost, the solution involves singular value shrinkage [1]
- Therefore, to find global solution, we iterate over possible schemes and re-use steps for fixed scheme.

Low-rank with scheme selection: Experiments





Insights:

- selecting the scheme along with ranks outperforms rank selection
- the scheme 2 is selected more often (as optimal) per each layer

The software

The LC algorithm: pseudocode and software implementation

input training data and model with parameters $\mathbf{w}$

$\mathbf{w} \leftarrow \overline{\mathbf{w}} = \arg\min_{\mathbf{w}} L(\mathbf{w})$ pretrained model

$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}^{\mathrm{DC}} = \boldsymbol{\Pi}(\overline{\mathbf{w}})$ init compression

$\boldsymbol{\beta} \leftarrow \mathbf{0}$

for $\mu = \mu_0 < \mu_1 < \cdots < \infty$

  $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta}) - \frac{1}{\mu}\boldsymbol{\beta}\|^2$ L step

  $\boldsymbol{\Theta} \leftarrow \arg\min_{\boldsymbol{\Theta}} \frac{\mu}{2}\|\mathbf{w} - \frac{1}{\mu}\boldsymbol{\beta} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2 + \lambda C(\boldsymbol{\Theta})$ C step

  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \mu(\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta}))$ multipliers step

  if $\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|$ is small enough **then** exit the loop

return $\mathbf{w}$, $\boldsymbol{\Theta}$

```
class LCAlgorithm():
  # Housekeeping code skipped
  def run(self):
    self.mu = 0
    self.c_step(step_number=0)
    for i, mu in enumerate(self.mu_schedule):
      self.mu = mu
      self.l_step(i) # user defined
      self.c_step(i) # library call
      self.multipliers_step()
```

The LC software

- Written in `python` using `NumPy` and `PyTorch`
- L step We hand off the L step to the user through the lambda functions.

```
def my_l_step(model, lc_penalty, args**):
    # ...
    loss = model.loss(out_, target_) + lc_penalty()
    loss.backward()
    optimizer.step()
    # ...
```

- C step Many compression are implemented, and you can chose any. If desired, you can add your own compression too by extending the `CompressionTypeBase`: class.

```
class ScaledBinaryQuantization(CompressionTypeBase):
    def compress(self, data):
        a = np.mean(np.abs(data))
        quantized = 2 * a * (data > 0) - a
        return quantized
```

The LC software: currently implemented compressions

Type	Forms
Quantization	Adaptive Quantization into $\{c_1, c_2, \dots c_K\}$ Binarization into $\{-1, 1\}$ and $\{-c, c\}$ Ternarization into $\{-c, 0, c\}$
Pruning	ℓ_0-constraint (s.t., $\|\mathbf{w}\|_0 \leq \kappa$) ℓ_1-constraint (s.t., $\|\mathbf{w}\|_0 \leq \kappa$) ℓ_0-penalty ($\alpha\|\mathbf{w}\|_0$) ℓ_1-penalty ($\alpha\|\mathbf{w}\|_1$)
Low-rank	Low-rank compression to a given rank Low-rank with *automatic* rank selection for FLOPs reduction Low-rank with *automatic* rank selection for storage compression Low-rank with *device-targeted* compression
Additive Combinations	Quantization + Pruning Quantization + Low-rank Pruning + Low-rank Quantization + Pruning + Low-rank

The LC software: the ease of exploration

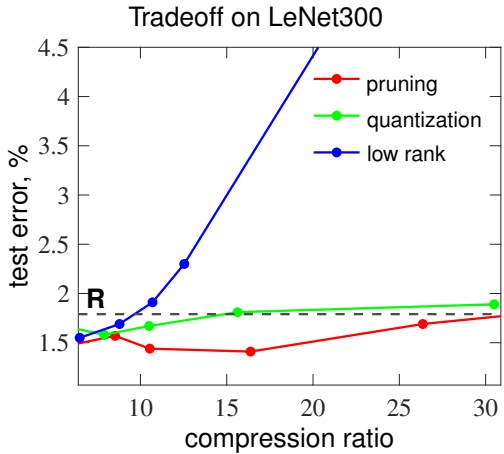
- Mix-and-match through compression tasks. We structured the software in such way that any compression can be applied to any part of the model, and you can mix them as well!
  For example, the following semantics:

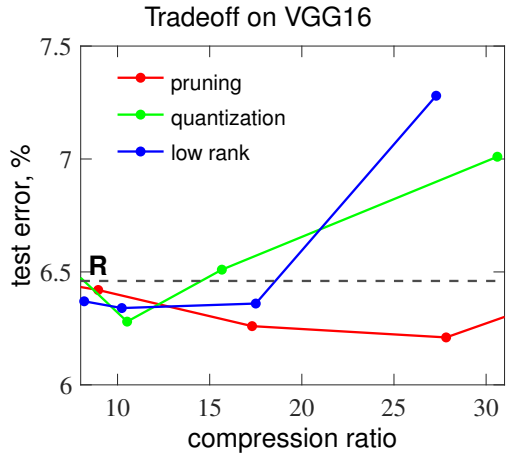
  (layer 1, layer 3) $\rightarrow$ adaptive quantization $k = 6$
  (layer 2) $\rightarrow$ low-rank with $r = 3$

  Translates into the following code:

```
from lc.torch import ParameterTorch as P, AsVector, AsIs
compression_tasks = {
  P([l1.weight, l3.weight]): (AsVector, AdaptiveQuantization(k=6)),
  P(l2.weight):              (AsIs,     LowRank(target_rank=3))
}
```

Example: Apples-to-apples comparison between compressions



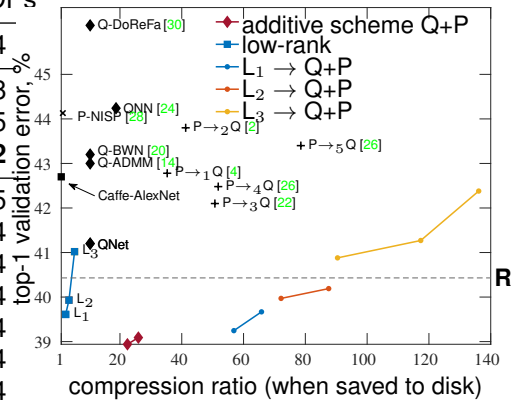


See more on it in [9]

Example: Additive compressions to achieve smallest AlexNets

The frameworks and software allows easy exploration of new compressions. For example, how about additive combination of quantization and pruning?

	Model	top-1	size, MB	MFLOPs
	Caffe-AlexNet Jia et al. [11]	42.70	243.5	724
ours	$L_1 \rightarrow$ Q (1-bit) + P (0.25M)	**39.67**	3.7	228
ours	$L_2 \rightarrow$ Q (1-bit) + P (0.25M)	40.19	2.8	185
ours	$L_3 \rightarrow$ Q (1-bit) + P (0.25M)	41.27	**2.1**	**152**
	AlexNet-QNN of Wu et al. [24]	44.24	13.0	175
	$P \rightarrow_1 Q$ of Han et al. [4]	42.78	6.9	724
	$P \rightarrow_2 Q$ of Choi et al. [2]	43.80	5.9	724
	$P \rightarrow_3 Q$ of Tung and Mori [22]	42.10	4.8	724
	$P \rightarrow_4 Q$ of Yang et al. [26]	42.48	4.7	724
	$P \rightarrow_5 Q$ of Yang et al. [26]	43.40	3.1	724
	filter pruning of Li et al. [17]	43.17	232.0	334



More on it in [10]

Conclusion and future work

- We have presented the Learning-Compression algorithm: and showed how it can be applied to the problems low-rank compression for NNs.
- The resulting algorithms require alternation of two steps: learning (L) step involving model loss and dataset and compression (C) step involving actual compression problem.
- Our trained models achieve considerable amount of compression and are on-par or better than other approaches.
- During the course of my PhD studies, we have extended the LC framework to include following NN compressions:
  - quantization (arxiv, CVPR2021)
  - pruning (CVPR2018)
  - low-rank (CVPR2020, DCC2021, ICASSP2021, ICIP2021)
  - additive combinations of above (ECML2021)
  - comparison of pruning, quantization, and low-rank (IJCNN2021)
- Many extensions are possible . . .

Future work: New models

My work is primarily about neural networks for vision, but, other models are of interest:

- language models like LSTMs, Transformers:
  - a particular interest is in compression of (tall) softmax layers
  - heteregenous structure of such models imply selection of “best” compressions
- recommendation systems

General ML models:

- probabilistic models (e.g., MMs and HMMs)
- kernel methods (for both classification and regression)

A particular challenge with these are a special constraints: e.g., with HMM’s Gaussian emission matrices, weight must be psd

Future work: Cost-based selection mechanisms

We have demonstrated the power of compression cost function $C(\mathbf{W})$ that allows to select best ranks for size, FLOPs and runtime.

$$\min_{\mathbf{W},\boldsymbol{\Theta}} L(\mathbf{W}) + \lambda C(\boldsymbol{\Theta}) \quad \text{s.t.} \quad \mathbf{W} = \boldsymbol{\Delta}(\boldsymbol{\Theta})$$

There are a lot of extensions on this front:

- Design of new cost-functions targeting:
  - power-consumption in the data center/mobile-device/etc
  - layout and placement of model (e.g., store $\mathbf{W}$ or $\mathbf{W}^T$, run everything on CPU or GPU?)
  - utilization (cache misses, transfer speeds, loading times)
- Hardware-Compression co-design:
  - design hardware features optimized for specific compressions/models

Future work: Single entry for the world of model compression

Model compression is hard, but our LC framework, its extensions and software based on it allow us to drastically simplify it for users:

- Users only need to provide an L-step implementation (SGD-based) and choose required compressions from the library
- Not harder than training the original model

Due to its simplicity and competitive results, we believe it can be a single entry point to model compression:

- as `PyTorch` or `TensorFlow` but for model compression

To do so, the following is required:

- include new models/selection mechanisms/compressions/results
- advertise/popularize among community members
- include out-of-the-box support for “standard” features (framework bindings, pipelines, etc)
- create tutorials/workshops
- look for an industry support

The end. Thank you!

References

[1] J.-F. Cai, E. J. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM J. Optimization*, 20(4):1956–1982, 2010.

[2] Y. Choi, M. El-Khamy, and J. Lee. Towards the limit of network quantization. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*

[3] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan. Approximated oracle filter pruning for destructive CNN width optimization. In *Proc. of the 36th Int. Conf. Machine Learning (ICML 2019)*

[4] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*

[5] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proc. 16th Int. Conf. Computer Vision (ICCV'17)*

[6] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. 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)*

[7] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. In *Proc. 15th European Conf. Computer Vision (ECCV'18)*

[8] Y. Idelbayev and M. 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)*

[9] Y. Idelbayev and M. 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).*

[10] Y. Idelbayev and M. 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)*

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093 [cs.CV], June 20 2014.

[12] H. Kim, M. U. K. Khan, and C.-M. Kyung. Efficient neural network compression. In *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*

[13] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*

[14] C. Leng, H. Li, S. Zhu, and R. Jin. Extremely low bit neural network: Squeeze the last bit out with ADMM. In *Proc. of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*

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

[16] H. Li, A. Kadav, I. Durdanovic, and H. P. Graf. Pruning filters for efficient ConvNets. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*

[17] J. Li, Q. Qi, J. Wang, C. Ge, Y. Li, Z. Yue, and H. Sun. OICSR: Out-in-channel sparsity regularization for compact deep neural networks. In *2019 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'19)*

[18] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. 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)*

[19] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. 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)*

[20] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-net: ImageNet classification using binary convolutional neural networks. In *Proc. 14th European Conf. Computer Vision (ECCV'16)*

[21] C. Tai, T. Xiao, Y. Zhang, X. Wang, and W. E. Convolutional neural networks with low-rank regularization. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*

[22] F. Tung and G. Mori. CLIP-Q: Deep network compression learning by in-parallel pruning-quantization. In *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*

[23] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li. Coordinating filters for faster deep neural networks. In *Proc. 16th Int. Conf. Computer Vision (ICCV'17)*

[24] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*

[25] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi. Scaling for edge inference of deep neural networks. *Nature Electronics*, 1:216–222, Apr. 17 2018.

[26] H. Yang, S. Gui, Y. Zhu, and J. Liu. Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach. In *Proc. of the 2020 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20)*

[27] J. Ye, X. Lu, Z. Lin, and J. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *Proc. of the 6th Int. Conf. Learning Representations (ICLR 2018)*

[28] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. 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)*

[29] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian. Variational convolutional neural network pruning. In *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*

[30] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv:1606.06160, July 17 2016.

[31] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pages 815–886.

Additional slides

Easy exploration of compressions

Having an L-step implementation (you only need one), definition of compression is very simple:

quantize each layer with separate codebooks

```
compression_tasks = {
  Param(l1.weight): (AsVector, AdaptiveQuantization(k=2)),
  Param(l2.weight): (AsVector, AdaptiveQuantization(k=2)),
  Param(l3.weight): (AsVector, AdaptiveQuantization(k=2))
}
```

---

prune all but 5%

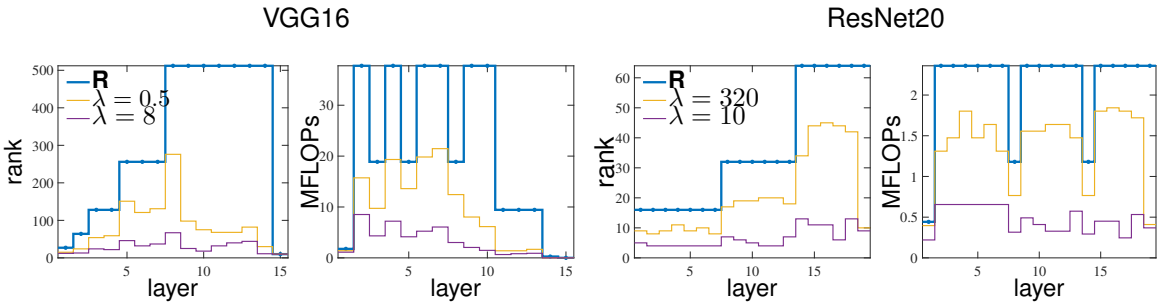
```
compression_tasks = {
  Param([l1.weight, l2.weight, l3.weights]):
    (AsVector, ConstraintL0Pruning(kappa=13310)) # 13310 =
    5%
}
```

---

prune first layer, low-rank to second, quantize third

```
compression_tasks = {
  Param(l1.weight): (AsVector, ConstraintL0Pruning(kappa
   =5000)),
  Param(l2.weight): (AsIs,     LowRank(target_rank=10))
  Param(l3.weight): (AsVector, AdaptiveQuantization(k=2))
}
```

Low-rank: experiments on CIFAR10 (selected ranks)



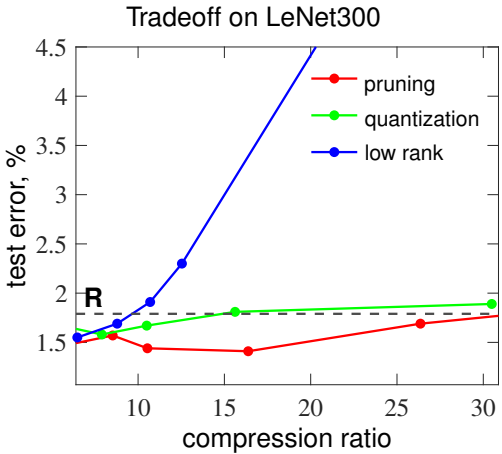
Ranks and FLOPs over layers for VGG16 and ResNet20 using our method for selected values of λ ($\times 10^{-4}$)

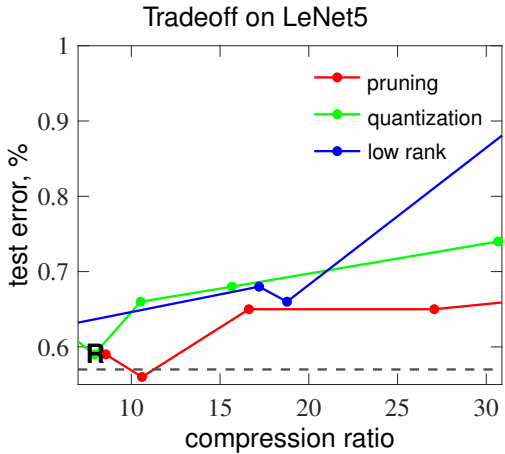
Low-rank with scheme selection: experiments on CIFAR10 (selected ranks & schemes)

The final selected ranks and reshaping schemes for some of the compressed VGG16 on CIFAR10

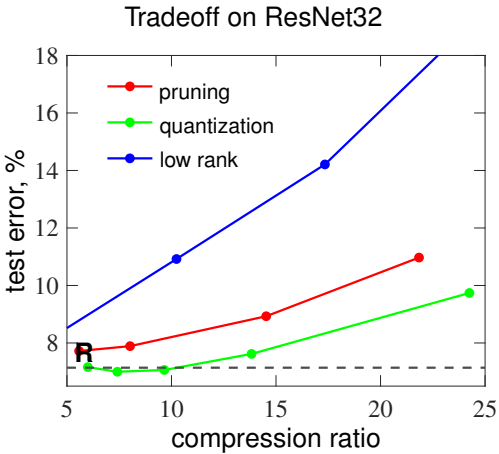
	test error	FLOPs	parameters	The selected scheme and rank over layers														
				1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\lambda = 2.0 \times 10^{-5}$	5.90%	156M	4.8M	$\mathcal{S}_1$ 16	$\mathcal{S}_2$ 32	$\mathcal{S}_2$ 71	$\mathcal{S}_2$ 97	$\mathcal{S}_3$ 116	$\mathcal{S}_2$ 238	$\mathcal{S}_2$ 263	$\mathcal{S}_3$ 254	$\mathcal{S}_2$ 292	$\mathcal{S}_2$ 172	$\mathcal{S}_2$ 122	$\mathcal{S}_2$ 99	$\mathcal{S}_2$ 105	31	11	9
$\lambda = 7.5 \times 10^{-5}$	5.97%	78M	2.5M	$\mathcal{S}_1$ 15	$\mathcal{S}_2$ 20	$\mathcal{S}_2$ 43	$\mathcal{S}_2$ 53	$\mathcal{S}_2$ 113	$\mathcal{S}_2$ 110	$\mathcal{S}_2$ 116	$\mathcal{S}_2$ 239	$\mathcal{S}_2$ 124	$\mathcal{S}_2$ 79	$\mathcal{S}_2$ 72	$\mathcal{S}_2$ 74	$\mathcal{S}_2$ 89	23	10	9

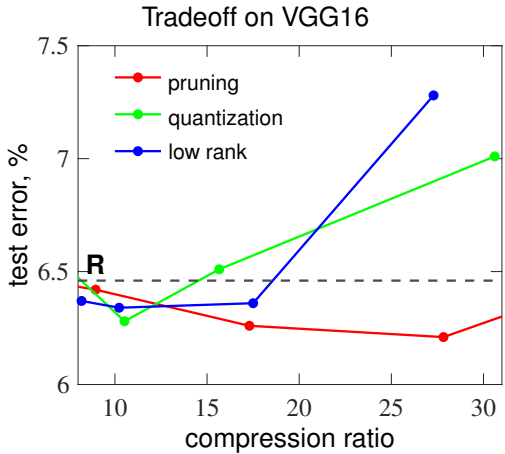
Quantization, pruning, and low-rank: Comparison on MNIST





Quantization, pruning, and low-rank: Comparison on CIFAR10





Why not to train smaller model in the first place?

We certainly can train smaller model (with fewer weights or layers) from scratch using SGD. However, it requires many trials to find best small network architecture.

Another option: if model (loss L and model function f) and decompression mapping are both differentiable, we can directly optimize:

$$\min_{\Theta} L(\mathbf{\Delta}(\Theta))$$

- Unfortunately, most of the compressions of interest are non differentiable
- Many tailored algorithms were proposed to bypass the non-differentiability of specific compression forms (BinaryConnect, STE)
- There is still the problem of selecting the best form of $\mathbf{\Delta}$ (say, what should be the ranks per layer?)

Generally, training a large model and then compressing it allows to peek into upper bound of the network performance, and adjust expectations wrt smaller, compressed model.

Convergence of the LC algorithm

The framework includes a large variety of compression problems that can be continuous (low-rank with given rank, ℓ_1 pruning) or combinatorial (rank selection, quantizaiton, ℓ_0 pruning)

Convergence of the LC alternation to a local minimizer can be established for:

- convex, differentiable cases (quite generally)
- nonconvex, fully differential cases (more restrictive results)

However, most interesting cases are those with nonconvex losses and non-differentiable constraints (quantization, ℓ_0 pruning). For these problems:

- convergence results are of general interest and is an open problem
- have flavor of k-means: algorithm stops at a point of no further improvement
- empirically, algorithm performs very well