Automated pipeline for neural network compression

Julia Gusak[1], *Maksym Kholiavchenko*[2], *Evgeny Ponomarev*[1], *Larisa Markeeva*[1], *Philip Blagoveschensky*[1], *Andrzej Cichocki*[1], *Ivan Oseledets*[1]

[1]Skolkovo Institute of Science and Technology, Russia

[2]Innopolis University, Russia

Contact e-mail: y.gusak@skoltech.ru

ICCV Low-Power Computer Vision Workshop, Seoul
October 28, 2019

Overview

Motivation

Modern neural networks (NN)

- Contain tens of millions of parameters
- Often cannot be efficiently deployed on embedded systems and mobile devices due to their computational power and memory limitations.
- Most expensive operations in CNNs are the convolution and multiplication in the convolutional and fully connected layers.

Motivation

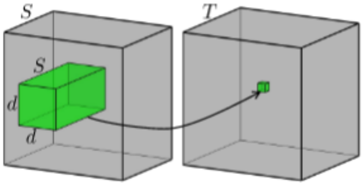
The main idea underlying the low-rank compression methods is that

- Modern CNNs are overparameterized and their weight tensors lie in a low-rank subspace.
- As a result, it is possible to compress a weight tensor by approximating it using different types of tensor decompositions, such as CP (canonical polyadic), Tucker, and other decompositions.
- This naturally reduces the number of parameters and operations in CNNs.

Motivation

For a convolutional layer with input of size $H \times W \times S$ and kernel (weight tensor) of size $d \times d \times T \times S$ number of

- parameters: $O(d^2ST)$
- operations: $O(HWd^2ST)$



Source: https://arxiv.org/pdf/1412.6553.pdf
Figure: Convolutional layer.

Reducing the number of parameters in CNNs is a common trick to accelerate inference time and at the same time reduce power usage and network memory.

Tensor decompositions

- $\underline{X}_{ijk} \cong \sum_{r=1}^{R} \lambda_r a_{ir} b_{jr} c_{kr}$ (1)

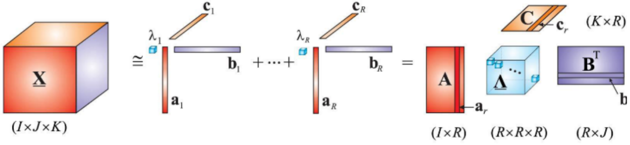


Figure: rank-R CP decomposition of 3D tensor

- $\underline{X}_{ijk} \cong \sum_{r_1}^{R_1} \sum_{r_2}^{R_2} \sum_{r_3}^{R_3} g_{r_1 r_2 r_3} b^{(1)}_{ir_1} b^{(2)}_{jr_2} b^{(3)}_{kr_3}$ (2)

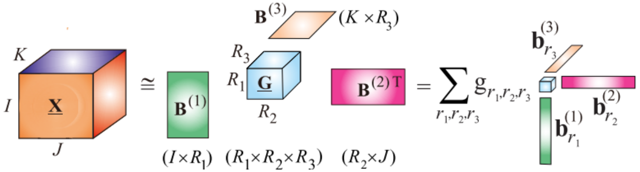
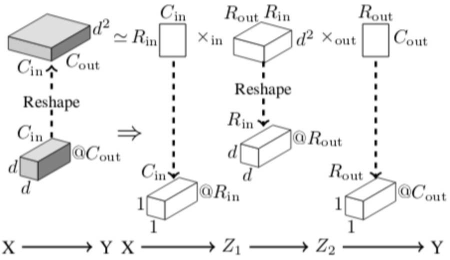


Figure: rank-(R_1, R_2, R_3) Tucker decomposition of 3D tensor

NN compression: compression step (Tucker-2 example)



- Top row: an approximation of a 3D weight tensor with a low-rank tensor, which can be represented in Tucker-2 format ($\times_{\text{in}} \times_{\text{out}}$ denote multilinear products along channel dimensions).
- Bottom row: initial layer is replaced with a sequence of 3 conv layers.
  - parameters: $O(d^2 C_{\text{in}} C_{\text{out}}) \rightarrow O(C_{\text{in}} R_{\text{in}} + d^2 R_{\text{out}} R_{\text{in}} + C_{\text{out}} R_{\text{out}})$,
  - operations:
    $O(HWd^2 C_{\text{in}} C_{\text{out}}) \rightarrow O(HWC_{\text{in}} R_{\text{in}} + H'W'(d^2 R_{\text{out}} R_{\text{in}} + C_{\text{out}} R_{\text{out}}))$

NN compression: one-shot

Low-rank matrix and tensor approximations provide excellent compression of neural network layers in many cases (Lebedev et al. 2014, Kim et al. 2015). They are built on the same scheme:

- Compress pre-trained neural network (NN)
- Fine-tune NN

Drawbacks: compression with a significant loss of accuracy, can yield to a bad initial approximation for further fine-tuning.

NN compression: iterative

Multi-stage approach with gradual rank reduction allows to tackle the problem arised in one-shot approach.

While desired compression rate is not reached or automatically selected ranks have not stabilized, **repeat**:

- Compress neural network (NN)
- Fine-tune NN

Benefits: compressed representation allows to find a good initial approximation.

NN compression: iterative

Algorithm 1 Iterative low-rank approximation algorithm for automated network compression

Input: Pre-trained original model, M
Output: Fine-tuned compressed model, M^*.

1: $M^* \leftarrow M$
2: **while** desired compression rate is not attained or automatically selected ranks have not stabilized **do**
3: $R \leftarrow$ automatically selected ranks for low-rank tensor approximations of convolutional and fully-connected weight tensors.
4: $M \leftarrow$ (further) compressed model obtained from M by replacing layer weights with their rank-R tensor approximations.
5: $M^* \leftarrow$ fine-tuned model M.
6: **end while**

NN compression: automatic rank selection

- **Bayesian approach**.
  - Firstly, rank R is found via GAS of EVBMF (Global Analytic Solution of Empirical Variational Bayesian Matrix Factorization),
  - Secondly, a rank weakening is performed. Rank weakening is applied in order to maintain some redundancy in the decomposed tensor for further iterative compression.
- **Constant compression rate**.
  - Ranks for tensor approximations can be chosen based on parameter/FLOPs reduction rate that we want to achieve at each compression step. By choosing rank in such a way, speed-up ratio of each convolutional layer can be controlled.
- **Constant layer acccuracy drop**.

NN compression: further compression step (Tucker-2 ex.)

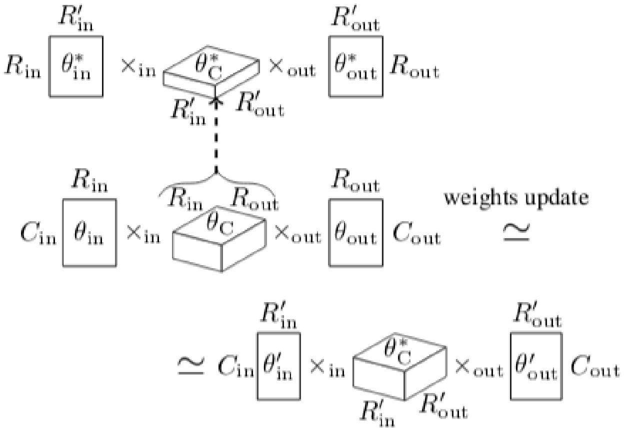


Figure: A low-rank approximation of a 3D tensor represented in Tucker-2 format

Results: one-shot vs iterative

Model	Size	CPU1	CPU2	GPU
AlexNet	4.90×	4.73×	4.55×	2.11×
YOLOv2	2.13×	2.07×	2.16×	1.62×
Tiny YOLOv2	2.30×	2.35×	2.28×	1.71×

Table: Results of iterative low-rank approximation. These tests were performed on CPUs of two different series and on GPU: Intel Core i5-7600K, Intel Core i7-7700K and NVIDIA GeForce GTX 1080 Ti, respectively.

Model	MUSCO	Tucker2-iter
AlexNet	-0.81	-4.2
YOLOv2	-0.19	-3.1
Tiny YOLOv2	-0.10	-2.7

Table: Quality drop after iterative compression and one-time compression. For classification networks metric is Δ Top-5 accuracy, for object detection - Δ mAP

Results: object detection (FasterRCNN with ResNet backbone)

Model	FLOPs	mAP
FASTER R-CNN C4 with RESNET-50 @ VOC2007+2012		
Used baseline	1.0×	75.0
Tucker2-iter (nx, 1.4)	1.17×	76.8(+1.8)
MUSCO(nx, 1.4, 2)	**1.39**×	**77.0(+2.0)**
MUSCO(nx, 1.4, 3)	**1.57**×	**75.4(+0.4)**
Tucker2-iter (nx, 3.16)	1.49×	75.0(+0.0)

Table: Comparison of Faster R-CNN (with ResNet-50 backbone) compressed models on Pascal VOC2007 dataset.

Results: image classification (ResNet on ILSVRC12)

Method	FLOPs	Δ top-1	Δ top-5
RESNET-18 @ ILSVRC12 dataset			
Network Slimming(Liu &al.,'17)	1.39	-1.77	-1.29
Low-cost Col. Layers(Dong &al.,'17)	1.53	-3.65	-2.3
Channel Gating NN(Hua &al., '18)	1.61	-1.62	-1.03
Filter Pruning(Li &al.,'17)	1.72	-3.18	-1.85
Discr.-aware Ch.Pr.(Zhuang &al.,'18)	1.89	-2.29	-1.38
FBS(Gao &al.,'18)	1.98	-2.54	-1.46
MUSCO (Ours)	**2.42**	**-0.47**	**-0.30**

Other compression techniques

- Quantization
- Pruning (layer/structural/fine-grained)
- Knowledge distillation
- Architecture search

Low-rank based compression method can be combined with the above techniques.

Link between deep learning architectures and tensor decompositions

Decomposing convolutional layer with *dxd* spatial kernel we obtain the following architecture units

- Tucker-2 decomposition - 1x1, dxd, 1x1 convolutions (ResNet block)
- CP decomposition - 1x1, depth-wise separable convolutions (MobileNet block)
- Block term decomposition - ResNext block

Further research Exploit the link between tensor decompositions and modern deep learning networks to construct effective architectures for the variety of applications.

Python packages: musco-pytorch, musco-tf

Installation

```
pip install musco-pytorch
```

Quick Start

```
from torchvision.models import resnet50
from from flopco import FlopCo
from musco.pytorch import CompressorVBMF, CompressorPR, CompressorManual

model = resnet50(pretrained = True)
model_stats = FlopCo(model, device = device)

compressor = CompressorVBMF(model,
                            model_stats,
                            ft_every=5,
                            nglobal_compress_iters=2)
while not compressor.done:
    # Compress layers
    compressor.compression_step()

    # Fine-tune compressed model.

compressed_model = compressor.compressed_model
```

Conclusion

- We propose an iterative low-rank approximation algorithm to efficiently compress neural networks that outperforms non-iterative methods for the desired accuracy.
- We introduce a method for automatic tensor rank selection for tensor approximations performed at each compression step.
- We validate and demonstrate the high efficiency of our approach in a series of extensive computational experiments for object detection and classification problems.
- **Preprint**: https://arxiv.org/abs/1903.09973
- **Source code** (PyTorch/TensorFlow): https://github.com/musco-ai

Thank you for your attention!