Tensor Methods for Neural Networks Speed-Up and Compression

Julia Gusak
y.gusak@skoltech.ru

Skolkovo Institute of Science and Technology, Moscow

July, 2021

Neural Network Compression: Motivation

- Most state of the art deep neural networks are overparameterized and exhibit a high computational cost.
- Often they cannot be efficiently deployed on embedded systems and mobile devices.
- Acceleration of pre-trained networks are usually achieved through structural pruning/sparcification, low-rank approximation and quantization.

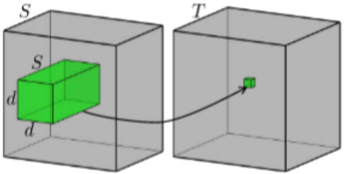
Recent Research

1. Low-rank tensor approximation of weight tensors to speed up and compress pre-trained NNs:
   - multi-stage compression (ICCVW, 2019, link),
   - stable low-rank approximation(ECCV, 2020, link).
2. Dimensionality reduction of activations (layers' outputs) to speed up and compress pre-trained NNs:
   - faster NNs using maximum volume algorithm (Computational Mathematics and Mathematical Physics Journal, 2021, link),
   - smaller NNs using active subspaces (SIAM Journal on Mathematics of Data Science, 2020, link).

NNs Compression via Weight Approximation: 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: $O(d^2 ST)$
- operations: $O(HWd^2 ST)$



Source: https://arxiv.org/pdf/1412.6553.pdf

Figure: Convolutional layer.

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

Tensor Decompositions for Weight Approximation

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

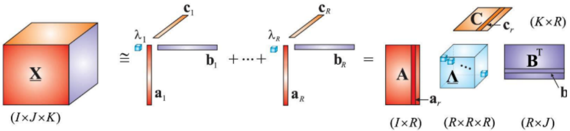


Figure: rank-R CP decomposition of 3D tensor (source: http://arxiv.org/abs/1609.00893)

Tensor Decompositions for Weight Approximation

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

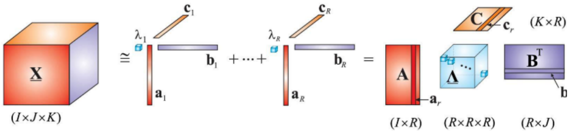


Figure: rank-R CP decomposition of 3D tensor (source: http://arxiv.org/abs/1609.00893)

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

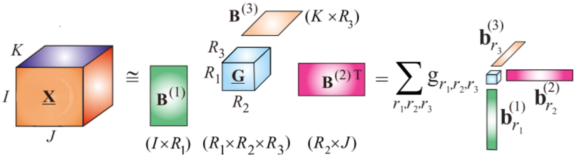
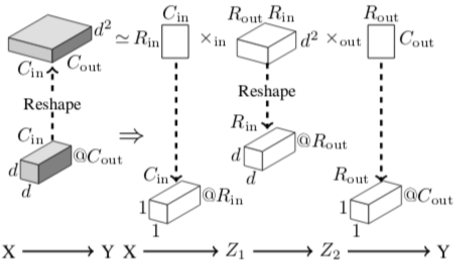


Figure: rank-(R_1, R_2, R_3) Tucker decomposition of 3D tensor

Layer Compression via Weight Approximation



- **Top row**: low-rank approximation of 3D weight tensor.
  - Tucker: $O(d^2 C_{\text{in}} C_{\text{out}}) \to O(C_{\text{in}} R_{\text{in}} + d^2 R_{\text{out}} R_{\text{in}} + C_{\text{out}} R_{\text{out}})$ parameters,
  - CP: $O(d^2 C_{\text{in}} C_{\text{out}}) \to O\Big(R(C_{\text{in}} + d^2 + C_{\text{out}})\Big)$ parameters, $R = R_{out} = R_{in}$.
- **Bottom row**: initial layer is replaced with a sequence of layers.
  - Tucker: middle convolution is standard.
  - CP: middle convolution is depth-wise.

NN Compression via Weight Approximation: One-Stage

NN compression via low-rank tensor/matrix approximations of weight tensors is usually built on the following one-stage scheme:

- Compress a pre-trained neural network. For each layer do
  - Extract a convolutional kernel.
  - Decompose it into factors.
  - Replace initial layer by a sequence of layers with factors as kernels.
  - Calibrate NN statistics.
- Fine-tune NN.

Drawbacks:

- significant loss of accuracy for high compression rates,
- this yields a bad initialization for further fine-tuning.

Recent Research

1. Low-rank tensor approximation of weight tensors to speed up and compress pre-trained NNs:
   - multi-stage compression (ICCVW, 2019, link),
   - stable low-rank approximation(ECCV, 2020, link).
2. Dimensionality reduction of activations (layers' outputs) to speed up and compress pre-trained NNs:
   - faster NNs using maximum volume algorithm (Computational Mathematics and Mathematical Physics Journal, 2021, link),
   - smaller NNs using active subspaces (SIAM Journal on Mathematics of Data Science, 2020, link).

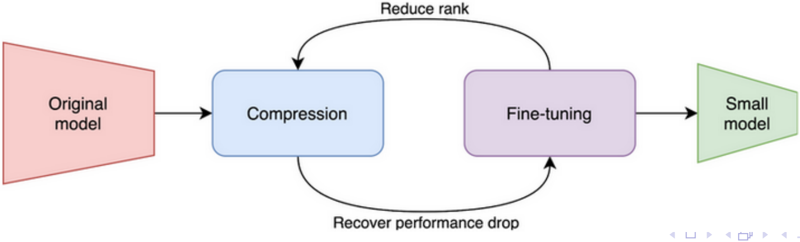
NN Compression: Multi-Stage

Multi-stage approach with gradual rank reduction addresses the problem arised in one-stage approach.

While a desired compression rate is not reached or automatically selected ranks are not stabilized, repeat:

- Compress the neural network.
- Fine-tune the neural network.

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



NN Compression: Automated Rank Selection

- **Constant compression rate**.

$$\#params(R_{new}) = \frac{\#params(R)}{rate}.$$

- **Constant layer acccuracy drop**.

$$accuracy(R) - accuracy(R_{new}) < drop,$$

accuracy is computed before fine-tuning, *drop* is a maximum allowed accuracy decrease caused by one layer compression.

- **Bayesian approach**. For each channel dimension

$$R_{new} = R - factor \cdot (R - R_{EVBMF}),$$

R_{EVBMF} is found via global analytic solution of Empirical Variational Bayesian Matrix Factorization, $0 \leq factor \leq 1$, $R_{EVBMF} \leq R_{new} \leq R$.

NN Compression: Further Compression Step

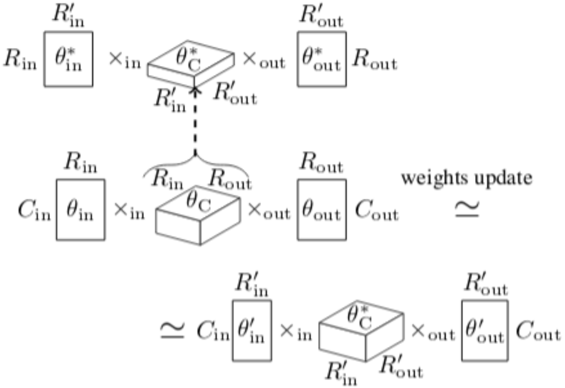


Figure: Compression of the factorized weight. Rank of the approximation is reduced from R to R'.

Results: MUlti-Stage COmpression

- MUlti-Stage COmpression outperforms one-stage compression on all tested
  - classification (AlexNet, VGG-16, ResNet-18, ResNet-50) and
  - object detection tasks (YOLOv2, TinyYOLO, FasterRCNN).
- Comparison with pruning approaches

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* (Our)**	**2.42**	**-0.47**	**-0.30**

- **Next:** How to stabilize fine-tuning with CP decomposed layers.

Recent Research

1. Low-rank tensor approximation of weight tensors to speed up and compress pre-trained NNs:
   - multi-stage compression (ICCVW, 2019, link),
   - stable low-rank approximation(ECCV, 2020, link).
2. Dimensionality reduction of activations (layers' outputs) to speed up and compress pre-trained NNs:
   - faster NNs using maximum volume algorithm (Computational Mathematics and Mathematical Physics Journal, 2021, link),
   - smaller NNs using active subspaces (SIAM Journal on Mathematics of Data Science, 2020, link).

CPD: Degeneracy

Standard CPD suffers from the presence of rank-one components that have relatively high Frobenius norms, but cancel each other.

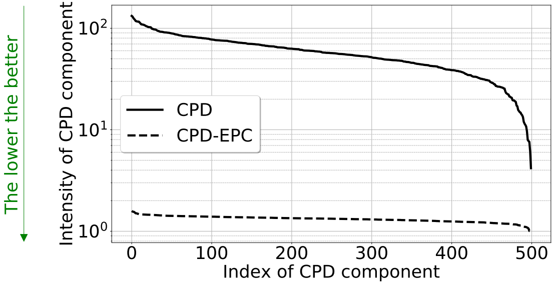


Figure: Intensity (Frobenius norm) for each rank-1 component from rank-500 CPD of a ResNet-18 weight. CPD-EPC states for the proposed decomposition.

CPD: Sensitivity

Sensitivity of the tensor $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ is a measure of factorized tensor norm change with respect to perturbations in individual factor matrices.

$$ss([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]) = \lim_{\sigma^2 \to 0} \frac{1}{R\sigma^2} \mathbb{E}\{\|\mathcal{T} - [\![\mathbf{A} + \delta\mathbf{A}, \mathbf{B} + \delta\mathbf{B}, \mathbf{C} + \delta\mathbf{C}]\!]\|_F^2,$$

where $\delta\mathbf{A}, \delta\mathbf{B}, \delta\mathbf{C} \sim \mathcal{N}(0, \sigma^2)$.

Sensitivity can be computed as

$$ss([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]) = \mathrm{tr}\{(\mathbf{A}^T\mathbf{A}) \circledast (\mathbf{B}^T\mathbf{B}) + (\mathbf{B}^T\mathbf{B}) \circledast (\mathbf{C}^T\mathbf{C}) + (\mathbf{A}^T\mathbf{A}) \circledast (\mathbf{C}^T\mathbf{C})\}$$

CPD-EPC

Proposed CPD-EPC is a CPD with minimal sensitivity

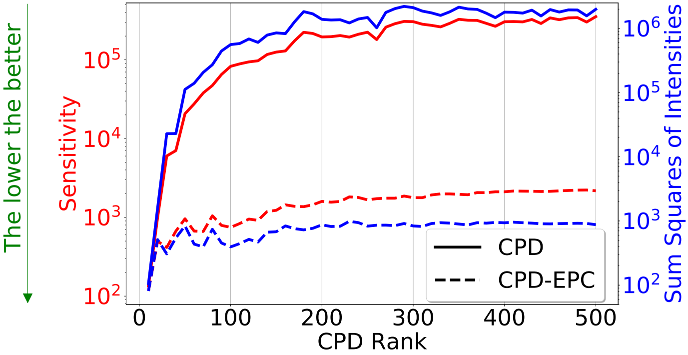
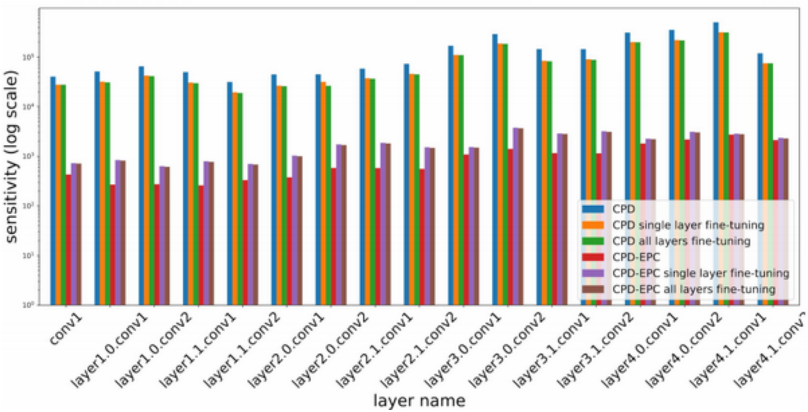


Figure: Sum of squares of the intensity and Sensitivity vs Rank of CPD.

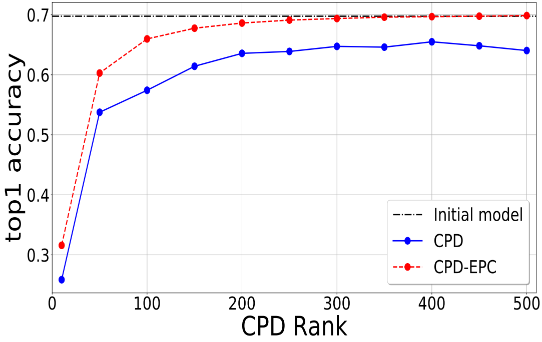
NN Compression via CPD: Sensitivity

Our method minimizes sensitivity of CPD making factorized layer stable during fine-tuning.



Results: NN Compression via CPD-EPC

CPD-EPC results in a significantly higher accuracy than the standard CPD.



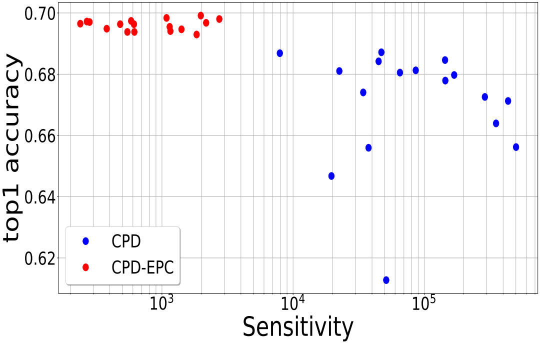


Figure: Evaluation of ResNet-18 with decomposed layer4.1.conv1 by CPD-EPC and original CPD after single layer fine-tuning, ILSVRC-12 dataset.

Results: NN Compression via CPD-EPC

Our method achieves the best performance in terms of compression-accuracy drop trade-off among all the considered results.

Table: Comparison of different model compression methods on ILSVRC-12 validation dataset. The baseline models are taken from Torchvision.

Model	**Method**	$\downarrow$ **FLOPs**	Δ **top-1**	Δ **top-5**
VGG-16	Asym. (Zhang&al., '16)	$\approx$ 5.00	-	-1.00
	TKD+VBMF(Kim&al.,'16)	4.93	-	-0.50
	Our (EPS[1]=0.005)	**5.26**	**-0.92**	**-0.34**
ResNet-18	*Channel Gating NN(Hua &al., '18)*	1.61	-1.62	-1.03
	Discr.-aware Ch.Pr.(Zhuang &al.,'18)	1.89	-2.29	-1.38
	FBS(Gao &al.,'18)	1.98	-2.54	-1.46
	MUSCO (Our'19)	2.42	-0.47	-0.30
	Our (EPS[1]=0.00325)	**3.09**	**-0.69**	**-0.15**
ResNet-50	**Our** (EPS[1]=0.0028)	**2.64**	**-1.47**	**-0.71**

Results: NN Compression via CPD-EPC

Table: Inference time and acceleration for ResNet-50 on different platforms.

Platform	Model inference time	
	Original	Compressed
Intel® Xeon® Silver 4114 CPU 2.20 GHz	3.92 ± 0.02 s	2.84 ± 0.02 s
NVIDIA® Tesla® V100	102.3 ± 0.5 ms	89.5 ± 0.2 ms
Qualcomm® Snapdragon™ 845	221 ± 4 ms	171 ± 4 ms

Python package: musco-pytorch

MUSCO is a Python library for NNs compression via tensor/matrix approximation of weight tensors.

- **Supported layers**: convolutional (1D, 2D), fully-connected.
- **Supported decompositions**: SVD, different types of CPD, Tucker decomposition.
- **Supported rank selection**: manual, constant compression rate, Bayesian (VBMF).
- Supports multi-stage compression.
- **Source code**: `https://github.com/musco-ai/musco-pytorch/tree/develop`

Python package: musco-pytorch

Steps to perform model compression using MUSCO package.

- Load a pre-trained model.
- Compute model statistics.
- Define a model compression schedule.
- Create a Compressor.
- Compress.

Python package: musco-pytorch

```python
from flopco import FlopCo
from musco.pytorch import Compressor

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

compressor = Compressor(model,
                        model_stats,
                        ft_every=5,
                        nglobal_compress_iters=2,
                        config_type = 'vbmf')
```

```python
while not compressor.done:
    # Compress layers
    compressor.compression_step()
    # Fine-tune compressor.compressed_model
```

For detailed instructions check *README.md* and *docs* at
`https://github.com/musco-ai/musco-pytorch/tree/develop`

NNs Compression via Weight Approximation: Conclusion

- Tensor based NN speedup can be improved by using
  - Multi-stage instead of one-stage compression,
  - CPD-EPC instead of standard CPD.
- **Further research**:
  - Joint low-rank tensor approximation and quantization for model compression.
  - Architectures that have both inputs and weights represented in a factorized format (potentially useful for efficient processing of very large models).
  - Robust tensorized architectures.

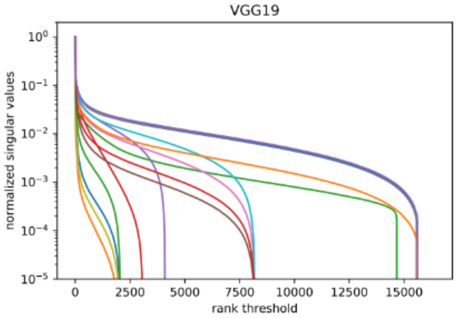
NNs Compression via Weight Approximation: Links with Efficient NNs

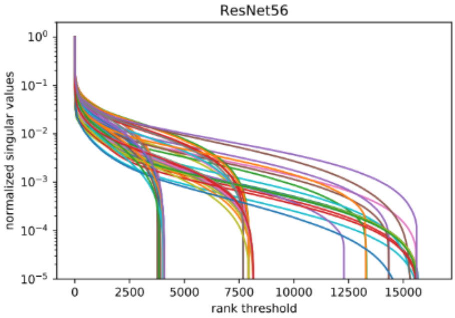
- There is a tight link between efficient DL blocks and layers that arise after applying different tensor decompositions to the standard convolutional kernels.
  - CP decomposition $\rightarrow$ MobileNet block,
  - Tucker decomposition $\rightarrow$ ResNet Bottleneck block,
  - Block Term decomposition $\rightarrow$ ResNext block.
- Hence, neural architecture search might be considered as a search for optimal decomposition.

Recent Research

1. Low-rank tensor approximation of weight tensors to speed up and compress pre-trained NNs:
    - multi-stage compression (ICCVW, 2019, link),
    - stable low-rank approximation(ECCV, 2020, link).
2. Dimensionality reduction of activations (layers' outputs) to speed up and compress pre-trained NNs:
    - faster NNs using maximum volume algorithm(Computational Mathematics and Mathematical Physics Journal, 2021, link),
    - smaller NNs using active subspaces (SIAM Journal on Mathematics of Data Science, 2020, link).

Our method relies on the assumption that the **outputs** of some layers can be mapped to a low-dimensional space





Reduced-Order Modelling of Network (RON): Multi-Layer Perceptron

- ψ_k $(k = 1, \ldots, K)$ are non-decreasing element-wise activation functions (e.g., ReLU, ELU or Leaky ReLU)
- $\boldsymbol{z}_0$ is an input sample, which undergoes the following transformations

$$\boldsymbol{z}_1 = \psi_1(\boldsymbol{W}_1 \boldsymbol{z}_0), \ \boldsymbol{z}_2 = \psi_2(\boldsymbol{W}_2 \boldsymbol{z}_1), \ \ldots, \ \boldsymbol{z}_K = \boldsymbol{W}_K \boldsymbol{z}_{K-1},$$

  where $\boldsymbol{W}_k \in \mathbb{R}^{D_k \times D_{k-1}}$ is a weight matrix of the k-th layer
- The low-rank assumption for the first layer:

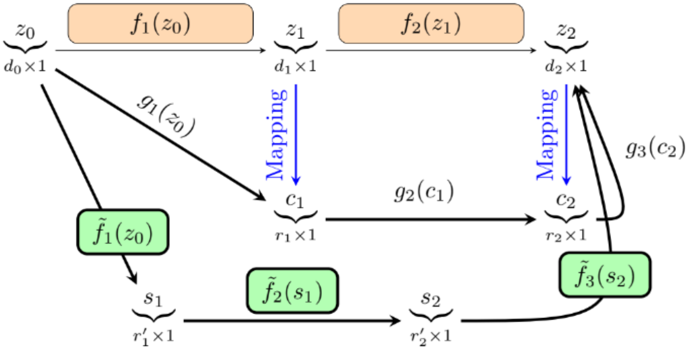
$$\boldsymbol{z}_1 \cong \boxed{\boldsymbol{V}_1 \boldsymbol{c}_1 \cong \psi_1(\boldsymbol{W}_1 \boldsymbol{z}_0)}$$

  where $\boldsymbol{c}_1$ is the embedding of $\boldsymbol{z}_1$
- We compute this embedding using the maximum volume algorithm:

$$\boldsymbol{c}_1 \cong (\boldsymbol{S}_1 \boldsymbol{V}_1)^\dagger \boldsymbol{S}_1 \psi_1 (\boldsymbol{W}_1 \boldsymbol{z}_0) = \underbrace{(\boldsymbol{S}_1 \boldsymbol{V}_1)^\dagger}_{R_1 \times P_1} \psi_1(\underbrace{\boldsymbol{S}_1 \boldsymbol{W}_1}_{P_1 \times D_1} \boldsymbol{z}_0),$$

  where $\boldsymbol{S}_1$ is a matrix which selects the relevant rows.

RON: Illustration



RON: From K-layer Network to a Faster $(K+1)$-layer Network

- We can compute $\boldsymbol{c}_2, \dots, \boldsymbol{c}_K$ using the same technique

$$\boldsymbol{c}_1 \cong \underbrace{(\boldsymbol{S}_1 \boldsymbol{V}_1)^\dagger}_{R_1 \times P_1} \psi_1(\underbrace{\boldsymbol{S}_1 \boldsymbol{W}_1}_{P_1 \times D_1} \boldsymbol{z}_0),$$

$$\boldsymbol{c}_k \cong \underbrace{(\boldsymbol{S}_k \boldsymbol{V}_k)^\dagger}_{R_K \times P_k} \psi_2(\underbrace{\boldsymbol{S}_k \boldsymbol{W}_k \boldsymbol{V}_{k-1}}_{P_k \times R_{k-1}} \boldsymbol{c}_{k-1}), \quad k = 2, \dots, K$$

$$\boldsymbol{z}_K \cong \boldsymbol{V}_K \boldsymbol{c}_K$$

- We get a $(K+1)$-layer neural network:

$$\boldsymbol{s}_1 \cong \psi_1(\underbrace{\boldsymbol{S}_1 \boldsymbol{W}_1}_{P_1 \times D_1} \boldsymbol{z}_0),$$

$$\boldsymbol{s}_k \cong \psi_k(\underbrace{\boldsymbol{S}_k \boldsymbol{W}_k \boldsymbol{V}_{k-1} (\boldsymbol{S}_{k-1} \boldsymbol{V}_{k-1})^\dagger}_{P_k \times P_{k-1}} \boldsymbol{s}_{k-1}), \quad k = 2, \dots, K$$

$$\boldsymbol{z}_K \cong \underbrace{\boldsymbol{V}_K (\boldsymbol{S}_K \boldsymbol{V}_K)^\dagger}_{D_k \times R_K} \boldsymbol{s}_K.$$

RON: Convolutional Networks

- **Convolution** is a linear transformation, and we treat it as a matrix-by-vector product, and we convert convolutions to fully-connected layers.
- **Batch normalization** can be merged with the dense layer for inference.
- **Maximum pooling** is a local operation, which typically maps 2×2 region into a single value — the maximum value in the given region. We manage this layer by taking 4 times more indices and by applying maximum pooling after sampling

RON: Residual Networks

We approximate the output of each branch and the result as follows

$$\boldsymbol{V}\boldsymbol{c} \cong \psi\left(\boldsymbol{V}_1\boldsymbol{c}_1 + \ldots + \boldsymbol{V}_k\boldsymbol{c}_k\right).$$

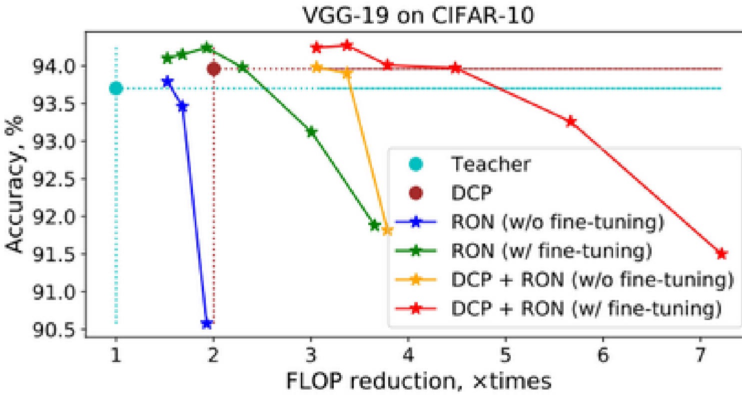
If $\boldsymbol{S}$ is a sampling matrix for matrix $\boldsymbol{V}$, the embedding $\boldsymbol{c}$ is computed as

$$\boldsymbol{c} \cong (\boldsymbol{S}\boldsymbol{V})^{\dagger}\,\psi\left(\boldsymbol{S}\boldsymbol{V}_1\boldsymbol{c}_1 + \ldots + \boldsymbol{S}\boldsymbol{V}_k\boldsymbol{c}_k\right).$$

The rest steps of residual networks acceleration are the same as for the standard multilayer perceptron.

RON: Results

Accuracy depending on FLOP reduction for models accelerated using Reduced-Order modelling of Neural Networks (RON).



RON: Remarks

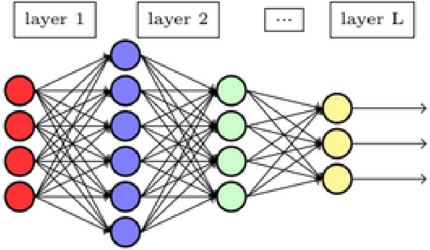
- The method can be applied on top of other acceleration methods and process the majority of popular network architectures.
- The resulting network is a simple multi-layer perceptron.
- In general, this method is for acceleration, not for compression.

Further research: build a faster network as a convolutional one, that will require keeping high-order structure of layers' outputs when performing dimesionality reduction.

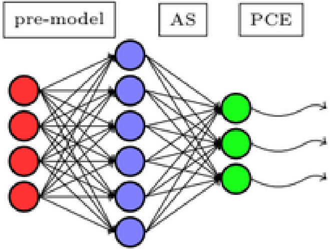
Recent Research

1. Low-rank tensor approximation of weight tensors to speed up and compress pre-trained NNs:
   - multi-stage compression (ICCVW, 2019, link),
   - stable low-rank approximation(ECCV, 2020, link).
2. Dimensionality reduction of activations (layers' outputs) to speed up and compress pre-trained NNs:
   - faster NNs using maximum volume algorithm (Computational Mathematics and Mathematical Physics Journal, 2021, link),
   - smaller NNs using active subspaces (SIAM Journal on Mathematics of Data Science, 2020, link).

NNs Compression using Active Subspaces



(a) A deep neural network



(b) The proposed ASNet

Active Subspace method uses the covariance matrix of gradient to find a projection matrix.
PCE states for polynomial chaos expansion.

Resume

- Tensors have a great potential to improve DL pipelines. Tensors
  - Incorporate higher-order correlations and multi-model data effectively.
  - Provide structural priors in DL.
  - Have been shown to impove DL applications (NNs speed-up/compression, one-shot learning, domain adaptation, incremental learning, fusion of features, etc.)
- Further research:
  - Joint low-rank approximation and quantization for model compression.
  - Architectures that have both inputs and weights represented in a factorized format.
  - Robust tensorized architectures.
  - Multi-modal feature fusion (e.g., images/video-point clouds, images-speech).
  - Combine reduced-order modeling technique (RON) and tensor methods to build a faster convolutional network.

References

- **Automated Multi-Stage Compression of neural networks.** Gusak J., Kholiavchenko M., Ponomarev E., Markeeva L., Cichocki A., Oseledets I. // ICCV Low-Power Computer Vision Workshop (2019). link
- **Stable Low-rank Tensor Decomposition for Compression of Convolutional Neural Network.** Phan A., Sobolev K., Sozykin K., Ermilov D., Gusak J., Tichavsky P., Glukhov V., Oseledets I., Cichocki A. // ECCV (2020). link
- **Reduced-Order Modeling of Deep Neural Networks.** Gusak J., Daulbaev T., Ponomarev E., Cichocki A., Oseledets I. // Computational Mathematics and Mathematical Physics Journal (2021). link
- **Active Subspace of Neural Networks: Structural Analysis and Universal Attacks.** Cui C., Zhang K., Daulbaev T., Gusak J., Oseledets I., Zhang Z. // SIAM Journal on Mathematics of Data Science, SIMODS (2020). link

Thank you!