CENG 5030
Energy Efficient Computing

Lecture 07: Low Rank Approximation

Bei Yu

(Latest update: March 8, 2021)

Spring 2021

Overview

Re-visit DNN Pruning

Low-Rank Approximation
- Singular Value Decomposition
- Tucker Decomposition
- CP-Decomposition

Unified Framework

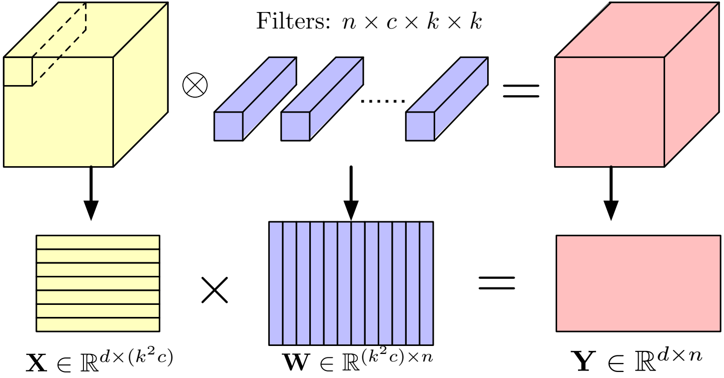
Overview

Re-visit DNN Pruning

Low-Rank Approximation
- Singular Value Decomposition
- Tucker Decomposition
- CP-Decomposition

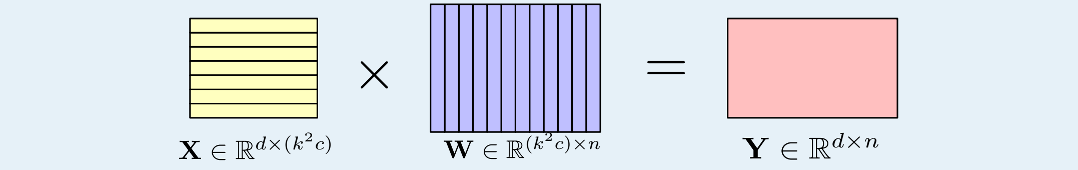
Unified Framework

Im2col (Image2Column) Convolution



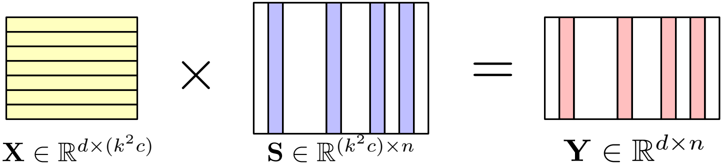
- Transform convolution to matrix multiplication
- Unified calculation for both convolution and fully-connected layers

Matrix Approximation or Matrix Regression?



- Matrix approximation: $\boldsymbol{W} \approx \boldsymbol{W'}$
- Matrix regression: $\boldsymbol{Y} = \boldsymbol{W} \cdot \boldsymbol{X} \approx \boldsymbol{W'} \cdot \boldsymbol{X}$

Compression Approach 1: Sparsity

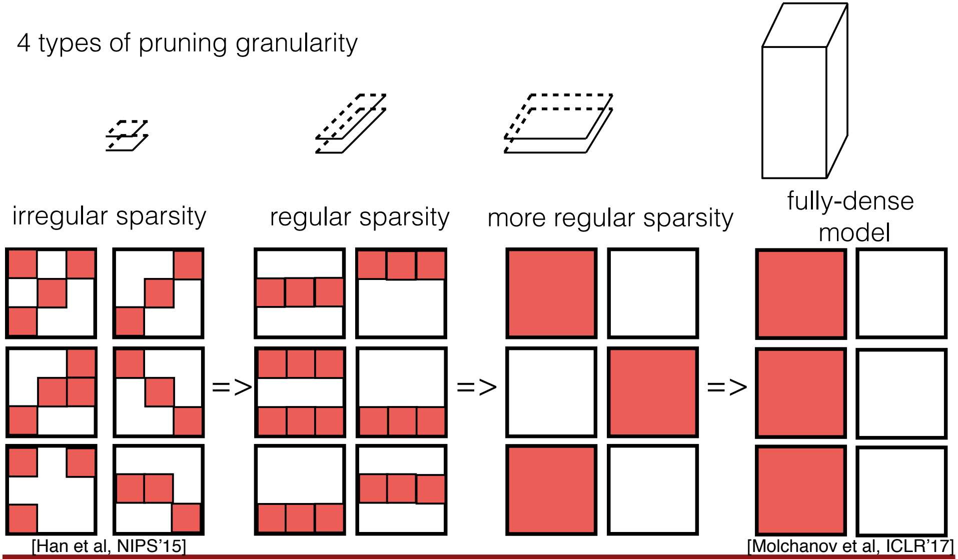


Sparse DNN

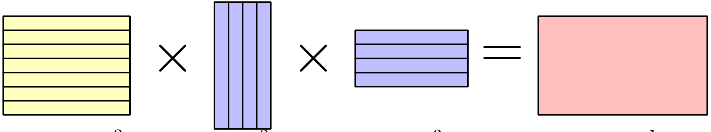
- *Sparsification*: weight pruning;
- *Compression*: compressed sparse format for storage;
- *Potential acceleration*: sparse matrix multiplication algorithm.

Exploring the Granularity of Sparsity that is Hardware-friendly

4 types of pruning granularity



Compression Approach 2: Low-Rank

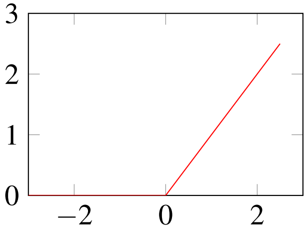


$\mathbf{X} \in \mathbb{R}^{d\times(k^2c)}$ $\mathbf{U} \in \mathbb{R}^{(k^2c)\times r}$ $\mathbf{V} \in \mathbb{R}^{1^2 r\times n}$ $\mathbf{Y} \in \mathbb{R}^{d\times n}$

Low-rank DNN

- *Low-rank approximation*: matrix decomposition or tensor decomposition.
- *Compression and acceleration*: less storage required and less FLOP in computation.

Non-linearity Approximation



ReLU

- Activation unit: `ReLU`
- Error more sensitive to positive response;
- Enlarge the solution space.

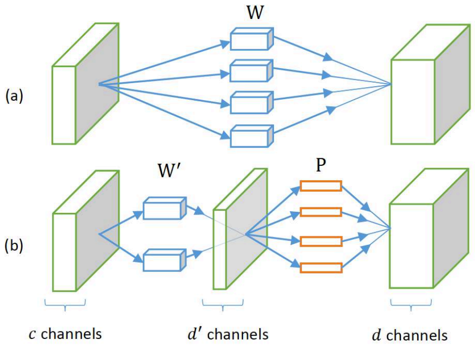
$$\min_{\boldsymbol{W}} \sum_{i=1}^{N} \|\boldsymbol{W}\boldsymbol{X}_i - \boldsymbol{Y}_i\|_F \rightarrow \min_{\boldsymbol{W}} \sum_{i=1}^{N} \|r(\boldsymbol{W}\boldsymbol{X}_i) - \boldsymbol{Y}_i\|_F$$

- $\boldsymbol{X}$: input feature map
- $\boldsymbol{Y}$: output feature map

Overview

Low Rank Approximation for Conv

- Layer responses lie in a low-rank subspace
- Decompose a convolutional layer with d filters with filter size $k \times k \times c$ to
  - A layer with d' filters ($k \times k \times c$)
  - A layer with d filter ($1 \times 1 \times d'$)



Zhang et al Efficient and Accurate Approximations of Nonlinear Convolutional Networks CVPR'15

Low Rank Approximation for Conv

speedup	rank sel.	Conv1	Conv2	Conv3	Conv4	Conv5	Conv6	Conv7	err. ↑ %
2×	no	32	110	199	219	219	219	219	1.18
2×	**yes**	32	83	182	211	239	237	253	**0.93**
2.4×	no	32	96	174	191	191	191	191	1.77
2.4×	**yes**	32	74	162	187	207	205	219	**1.35**
3×	no	32	77	139	153	153	153	153	2.56
3×	**yes**	32	62	138	149	166	162	167	**2.34**
4×	no	32	57	104	115	115	115	115	4.32
4×	**yes**	32	50	112	114	122	117	119	**4.20**
5×	no	32	46	83	92	92	92	92	6.53
5×	**yes**	32	41	94	93	98	92	90	**6.47**

Zhang et al Efficient and Accurate Approximations of Nonlinear Convolutional Networks CVPR'15

Low Rank Approximation for FC

Build a mapping from row / column indices of matrix $\boldsymbol{W} = [W(x,y)]$ to vectors $\boldsymbol{i}$ and $\boldsymbol{j}$: $x \leftrightarrow \boldsymbol{i} = (i_1, \ldots, i_d)$ and $y \leftrightarrow \boldsymbol{j} = (j_1, \ldots, j_d)$.

TT-format for matrix $\boldsymbol{W}$:

$$\boldsymbol{W}(i_1, \ldots, i_d; j_1, \ldots, j_d) = \boldsymbol{W}(x(\boldsymbol{i}), y(\boldsymbol{j})) = \underbrace{\boldsymbol{G}_1[i_1, j_1]}_{1\times r} \underbrace{\boldsymbol{G}_2[i_2, j_2]}_{r\times r} \ldots \underbrace{\boldsymbol{G}_d[i_d, j_d]}_{r\times 1}$$

Type	1 im. time (ms)	100 im. time (ms)
CPU fully-connected layer	16.1	97.2
CPU TT-layer	1.2	94.7
GPU fully-connected layer	2.7	33
GPU TT-layer	1.9	12.9

Novikov et al Tensorizing Neural Networks, NIPS'15

Singular Value Decomposition[1]

CONVOLUTIONAL NEURAL NETWORKS WITH LOW-RANK REGULARIZATION

Cheng Tai[1], Tong Xiao[2], Yi Zhang[3], Xiaogang Wang[2], Weinan E[1]
[1]The Program in Applied and Computational Mathematics, Princeton University
[2]Department of Electronic Engineering, The Chinese University of Hong Kong
[3]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor
{chengt,weinan}@math.princeton.edu; yeezhang@umich.edu
{xiaotong,xgwang}@ee.cuhk.edu.hk

[1]Cheng Tai et al. (2016). "Convolutional neural networks with low-rank regularization". In: *Proc. ICLR*.

Singular Value Decomposition

Contribution

- A new algorithm for computing the low-rank tensor decomposition
- A new method for training low-rank constrained CNNs from scratch
- Evaluation on large networks

Singular Value Decomposition

Pretrained CNN Approximation

- Convolution Calculation

$$\mathcal{F}_n(x,y) = \sum_{c=1}^{C} \sum_{x'=1}^{X} \sum_{y'=1}^{Y} \mathcal{Z}^c(x',y')\, \mathcal{W}_n^c(x-x', y-y')$$

- $\mathcal{W}_n \in \mathbb{R}^{d\times d\times C}$ to represent the n -th filter. $\mathcal{Z} \in \mathbb{R}^{X\times Y\times U}$ be the input feature map.
- An approximation of W

$$\tilde{\mathcal{W}}_n^c = \sum_{k=1}^{K} \mathcal{H}_n^k \left(\mathcal{V}_k^c\right)^T$$

where K is a hyper-parameter controlling the rank, $\mathcal{H} \in \mathbb{R}^{N\times 1\times d\times K}$ is the horizontal filter, $\mathcal{V} \in \mathbb{R}^{K\times d\times 1\times C}$ is the vertical filter (Notes: $\mathcal{H}_k^c$ and $\mathcal{V}_k^c$ are both vectors in $\mathbb{R}^d$). Both $\mathcal{H}$ and $\mathcal{V}$ are learnable parameters.

- Then the convolution becomes

$$\tilde{W}_n * \mathcal{Z} = \sum_{c=1}^{C} \sum_{k=1}^{K} \mathcal{H}_n^k \left(\mathcal{V}_k^c\right)^T * \mathcal{Z}^c = \sum_{k=1}^{K} \mathcal{H}_n^k * \left(\sum_{c=1}^{C} \mathcal{V}_k^c * \mathcal{Z}^c\right)$$

Singular Value Decomposition

Complexity Analysis

- Standard Convolution Complexity: $O(d^2NCXY)$ operations
- Approximation Scheme Complexity
  The computational cost associated with the vertical filters is $O(dKCXY)$ and with horizational fileters is $O(dNKXY)$, a total computational cost is $O(dK(N+C)XY)$
- If $K < \frac{dNC}{N+C}$, acceleration can be achieved

Singular Value Decomposition

Approximate Parameters H and V

- Minimizing the objective function

$$E_1(\mathcal{H},\mathcal{V}) := \sum_{n,c} \left\| \mathcal{W}_n^c - \sum_{k=1}^K \mathcal{H}_n^k \left(\mathcal{V}_k^c\right)^T \right\|_F^2$$

- Theorem: Define the following bijection that maps a tensor to a matrix $\mathcal{T} : \mathbb{R}^{C\times d\times d\times N} \mapsto \mathbb{R}^{Cd\times dN}$, tensor element (i_1, i_2, i_3, i_4) maps to (j_1, j_2) , where

$$j_1 = (i_1 - 1)\, d + i_2, \quad j_2 = (i_4 - 1)\, d + i_3$$

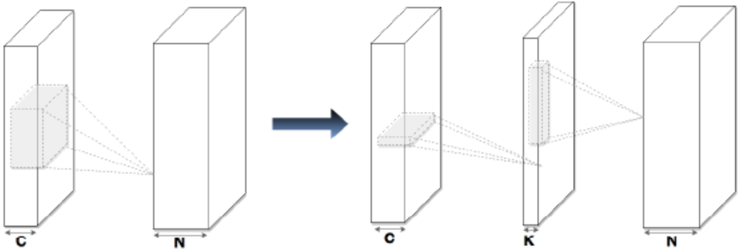
Define $W := \mathcal{T}[\mathcal{W}]$. Let $W = UDQ^T$ be the singular Value Decomposition (SVD) of W. Let

$$\hat{\mathcal{V}}_k^c(j) = U_{(c-1)d+j,k}\sqrt{D_{k,k}}$$
$$\hat{\mathcal{H}}_n^k(j) = Q_{(n-1)d+j,k}\sqrt{D_{k,k}}$$

then $(\hat{\mathcal{H}}, \hat{\mathcal{V}})$ is a solution to minimizing the object function

Singular Value Decomposition

- The proposed parametrization for low-rank regularization.



Left: The original convolutional layer. Right: low-rank constraint convolutional layer with rank-K.

Singular Value Decomposition

Training Low-rank Constrained CNN From Scratch

- The effect of SVD Decomposition
  Each convolutional layer is parameterized as the composition of two convolutional layers,
- Exploding and vanishing gradients expecially for large networks
- Batch Normalition can handle this problem
  (Recall the theory of Batch Normalization)

Tensor: Canonical Polyadic Decomposition

Tensor: Tucker Decomposition

Tensor: Tensor Train Decomposition

Tucker Decomposition[2]

COMPRESSION OF DEEP CONVOLUTIONAL NEURAL NETWORKS FOR FAST AND LOW POWER MOBILE APPLICATIONS

Yong-Deok Kim[1], Eunhyeok Park[2], Sungjoo Yoo[2], Taelim Choi[1], Lu Yang[1] & Dongjun Shin[1]

[1]Software R&D Center, Device Solutions, Samsung Electronics, South Korea
{yd.mlg.kim, tl.choi, lu2014.yang, d.j.shin}@samsung.com

[2]Department of Computer Science and Engineering, Seoul National University, South Korea
{canusglow, sungjoo.yoo}@gmail.com

[2] Yong-Deok Kim et al. (2016). "Compression of deep convolutional neural networks for fast and low power mobile applications". In: *Proc. ICLR*.

Tucker Decomposition

Contribution

- Propose a one-shot whole network compression scheme which consists of simple three steps: (1) rank selection, (2) low-rank tensor decomposition, and (3) fine-tuning.
- Tucker decomposition (Tucker, 1966) with the rank determined by a global analytic solution of variational Bayesian matrix factorization is applied on each kernel tensor.

Tucker Decomposition

Kernel Tensor Approximation

- Convolution Calculation

$$\mathcal{Y}_{h',w',t} = \sum_{i=1}^{D}\sum_{j=1}^{D}\sum_{s=1}^{S}\mathcal{K}_{i,j,s,t}\mathcal{X}_{h_i,w_j,s}$$

$$h_i = (h'-1)\,\Delta + i - P \text{ and } w_j = (w'-1)\,\Delta + j - P$$

  where $\mathcal{K}$ is a 4-way kernel tensor of size $D \times D \times S \times T$, δ is stride, and P is zero-padding size

- Tucker Decomposition:The rank-$(R_1; R_2; R_3; R_4)$ Tucker decomposition of 4-way kernel tensor K has the form:

$$\mathcal{K}_{i,j,s,t} = \sum_{r_1=1}^{R_1}\sum_{r_2=1}^{R_2}\sum_{r_9=1}^{R_3}\sum_{r_4=1}^{R_4}\mathcal{C}'_{r_1,r_2,r_3,r_4}U^{(1)}_{i,r_1}U^{(2)}_{j,r_2}U^{(3)}_{s,r_3}U^{(4)}_{t,r_4}$$

  where $\mathcal{C}'$ is a core tensor of size $R_1 \times R_2 \times R_3 \times R_4$ and $U^{(1)}, U^{(2)}, U^{(3)}$, and $U^{(4)}$ are factor matrices of sizes $D \times R_1, D \times R_2, S \times R_3$, and $T \times R_4$, respectively.

Tucker Decomposition

Tucker Decomposition

- Every mode does not have to be decomposed(e.g. For example, we do not decompose mode-1 and mode-2 which are associated with spatial dimensions because they are already quite small).
- Under this variant called Tucker-2 decomposition, the kernel tensor is decomposed to:

$$\mathcal{K}_{i,j,s,t} = \sum_{r_0=1}^{R_0} \sum_{r_4=1}^{R_4} \mathcal{C}_{i,j,r_3,r_4} U^{(3)}_{s,r_0} U^{(4)}_{t,r_4}$$

  where $\mathcal{C}$ is a core tensor of size $D \times D \times R_3 \times R_4$
- With the approximation of kernel, the convolution is as following:

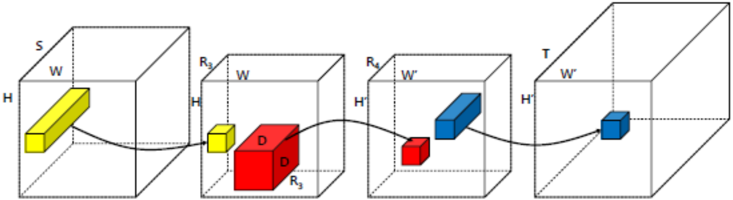
$$\mathcal{Z}_{h,w,r_3} = \sum_{s=1}^{S} U^{(3)}_{s,r_3} \mathcal{X}_{h,w,s}$$

$$\mathcal{Z}'_{h',w',r_4} = \sum_{i=1}^{D} \sum_{j=1}^{D} \sum_{r_0=1}^{R_0} \mathcal{C}_{i,j,r_3,r_4} \mathcal{Z}_{h_t,w_j,r_9}$$

$$\mathcal{Y}_{h',w',t} = \sum_{r_4=1}^{R_4} U^{(4)}_{t,r_4} \mathcal{Z}'_{h',w',r_4}$$

Tucker Decomposition

- Tucker-2 decompositions for speeding-up a convolution



- Complexity Analysis

$$M = \frac{D^2ST}{SR_3+D^2R_3R_4+TR_4} \text{ and } E = \frac{D^2STH'W'}{SR_3HW+D^2R_3R_4H'W'+TR_4H'W'}$$

M represents the compression ratio, E represents the speed-up ratio

Tucker Decomposition

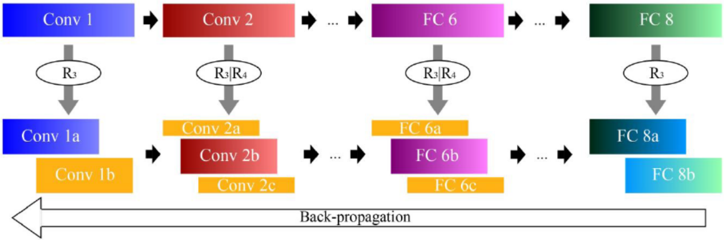
Rank Selection With Global Analytic VBMF

- Motivation: The rank-$(R_3; R_4)$ control the trade-off between performance (memory, speed, energy) improvement and accuracy loss.
- Method: variational Bayesian atrix factorization[3]
- Advantages: VBMF can automatically find noise variance, rank and even provide theoretical condition for perfect rank recovery

[3] Shinichi Nakajima et al. (2013). “Global analytic solution of fully-observed variational Bayesian matrix factorization”. In: *Journal of Machine Learning Research* 14.Jan, pp. 1–37.

Tucker Decomposition

- One-shot whole network compression scheme



Three parts: (1) rank selection with VBMF; (2) Tucker decomposition on kernel tensor; (3) fine-tuning of entire network.

- Notes:Tucker-2 decomposition is applied from the second convolutional layer to the first fully connected layers, and Tucker-1 decomposition to the other layers.

CP-Decomposition[4]

SPEEDING-UP CONVOLUTIONAL NEURAL NETWORKS USING FINE-TUNED CP-DECOMPOSITION

Vadim Lebedev[1,2], Yaroslav Ganin[1], Maksim Rakhuba[1,3], Ivan Oseledets[1,4], and Victor Lempitsky[1]

[1]Skolkovo Institute of Science and Technology (Skoltech), Moscow, Russia
[2]Yandex, Moscow, Russia
[3]Moscow Institute of Physics and Technology, Moscow Region, Russia
[4]Institute of Numerical Mathematics RAS, Moscow, Russia

[4] Vadim Lebedev et al. (2015). "Speeding-up convolutional neural networks using fine-tuned CP-decomposition". In: *Proc. ICLR*.

CP Decomposition

Method Overview

- Take a convolutional layer and decompose its kernel using CP-decomposition
- Fine-tune the entire network using backpropagation.

Advantages

- Ease of the decomposition implementation
- Ease of the CNN implementation
- Ease of fine-tuning
- Efficiency

CP-Decomposition

Principle

- A low-rank decomposition of a matrix A of size $n \times m$ with rank R is given by

$$A(i,j) = \sum_{r=1}^{R} A_1(i,r)A_2(j,r), \quad i = \overline{1,n}, \quad j = \overline{1,m}$$

- For a d-dimensional array A of size $n_1 \times \cdots \times n_d$ a CP-decomposition has the following form

$$A\left(i_1, \ldots, i_d\right) = \sum_{r=1}^{R} A_1\left(i_1, r\right) \ldots A_d\left(i_d, r\right)$$

  where the minimal possible R is called canonical rank.

- Profit we need to store only $\left(n_1 + \cdots + n_d\right) R$ elements instead of the whole tensor with $n_1 \ldots n_d$ elements.

Notes

- There is no finite algorithm for determining canonical rank of a tensor when $d > 2$
- Non-linear least squares (NLS) method is applied in this paper, which minimizes the L2-norm of the approximation residual (for a user-defined fixed R) using Gauss-Newton optimization.

CP-Decomposition

Kernel Tensor Approximation

- Convolution Calculation

$$V(x,y,t) = \sum_{i=x-\delta}^{x+\delta} \sum_{j=y-\delta}^{y+\delta} \sum_{s=1}^{S} K(i-x+\delta, j-y+\delta, s, t)U(i,j,s)$$

- $K(\cdot,\cdot,\cdot,\cdot)$ is a 4D kernel tensor of size $d \times d \times S \times T$ d is the spatial dimensions, S is input channels, T is output channels, while δ denotes "half-width" $(d-1)/2$
- Kernel Approximation

$$K(i,j,s,t) = \sum_{r=1}^{R} K^x(i-x+\delta, r)K^y(j-y+\delta, r)K^s(s,r)K^t(t,r)$$

- where $K^x(\cdot,\cdot), K^y(\cdot,\cdot), K^s(\cdot,\cdot), K^t(\cdot,\cdot)$ are the four components of the composition representing 2D tensors (matrices) of sizes $d \times R, d \times R, S \times R$, and $T \times R$ respectively.

Convolution Approximation

- Substitue the Kernel Approx to Conv

$$V(x,y,t) = \sum_{r=1}^{R} K^t(t,r)\left(\sum_{i=x-\delta}^{x+\delta} K^x(i-x+\delta,r)\left(\sum_{j=y-\delta}^{y+\delta} K^y(j-y+\delta,r)\left(\sum_{s=1}^{S} K^s(s,r)U(i,j,s)\right)\right)\right)$$

- Step by Step Calculation

$$U^s(i,j,r) = \sum_{s=1}^{S} K^s(s,r)U(i,j,s)$$

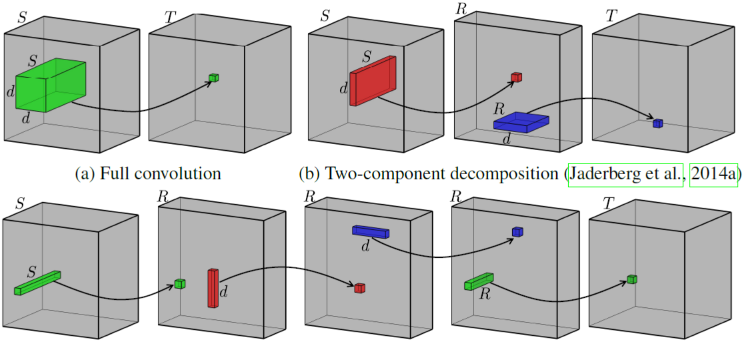
$$U^{sy}(i,y,r) = \sum_{j=y-\delta}^{y+\delta} K^y(j-y+\delta,r)U^s(i,j,r)$$

$$U^{\text{syx}}(x,y,r) = \sum_{i=x-\delta}^{x+\delta} K^x(i-x+\delta,r)U^{sy}(i,y,r)$$

$$V(x,y,t) = \sum_{r=1}^{R} K^t(t,r)U^{\text{syx}}(x,y,r)$$

CP-Decomposition

- Complexity Comparison



(a) Full convolution (b) Two-component decomposition (Jaderberg et al., 2014a)

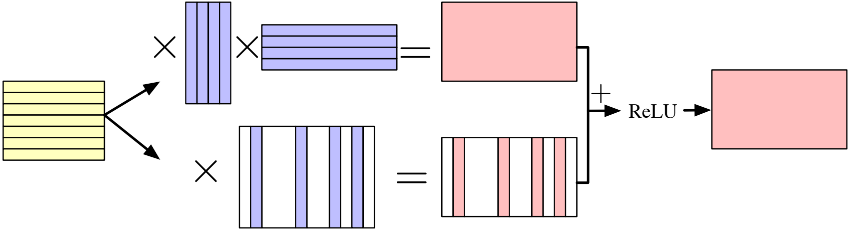
(c) CP-decomposition

Further Reading List

- Hao Zhou, Jose M Alvarez, and Fatih Porikli (2016). "Less is more: Towards compact cnns". In: *Proc. ECCV*, pp. 662–677
- Yihui He, Xiangyu Zhang, and Jian Sun (2017). "Channel Pruning for Accelerating Very Deep Neural Networks". In: *Proc. ICCV*
- Xiyu Yu et al. (2017). "On compressing deep models by low rank and sparse decomposition". In: *Proc. CVPR*, pp. 7370–7379

Overview

Proposed Unified Structure



- Simultaneous low-rank approximation and network sparsification;
- Non-linearity is taken into account.
- Acceleration is achieved with structured sparsity.

Formulation

Given a pre-trained network, the goal is to minimize the reconstruction error of the response in each layer after activation, using sparse component and low-rank component.

$$
\begin{aligned}
\min_{\boldsymbol{A},\boldsymbol{B}} \; & \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r((\boldsymbol{A}+\boldsymbol{B})\boldsymbol{X}_i)\|_F \,, \\
\text{s.t.} \; & \|\boldsymbol{A}\|_0 \le S, \\
& \text{rank}(\boldsymbol{B}) \le L.
\end{aligned}
$$

- $\boldsymbol{X}$: input feature map
- $\boldsymbol{Y}$: output feature map

Not easy to solve: l_0 minimization and rank minimization are NP-hard.

Relaxation

$$\min_{\boldsymbol{A},\boldsymbol{B}} \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r((\boldsymbol{A}+\boldsymbol{B})\boldsymbol{X}_i)\|_F^2 + \lambda_1 \|\boldsymbol{A}\|_{2,1} + \lambda_2 \|\boldsymbol{B}\|_*$$

- The l_0 constraint is relaxed by $l_{2,1}$ norm such that the zero elements in $\boldsymbol{A}$ appear column-wise;
- The rank constraint on $\boldsymbol{B}$ is relaxed by nuclear norm of $\boldsymbol{B}$, which is the sum of the singular values;
- Apply alternating direction method of multipliers (ADMM) to solve it;

Alternating Direction Method of Multipliers (ADMM)

Reformulating the problem with an auxiliary variable $\boldsymbol{M}$,

$$\begin{aligned}
&\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{M}} \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2 + \lambda_1 \|\boldsymbol{A}\|_{2,1} + \lambda_2 \|\boldsymbol{B}\|_* , \\
&\text{s.t. } \boldsymbol{A} + \boldsymbol{B} = \boldsymbol{M}.
\end{aligned}$$

Then the augmented Lagrangian function is

$$\begin{aligned}
&L_t(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{M}, \boldsymbol{\Lambda}) \\
&= \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2 + \lambda_1 \|\boldsymbol{A}\|_{2,1} + \lambda_2 \|\boldsymbol{B}\|_* + \langle \boldsymbol{\Lambda}, \boldsymbol{A} + \boldsymbol{B} - \boldsymbol{M} \rangle + \frac{t}{2} \|\boldsymbol{A} + \boldsymbol{B} - \boldsymbol{M}\|_F^2 ,
\end{aligned}$$

Alternating Direction Method of Multipliers (ADMM)

Iteratively solve with following rules. All of them can be solved efficiently.

$$\begin{cases} \boldsymbol{A}_{k+1} = \underset{\boldsymbol{A}}{\operatorname{argmin}}\ \lambda_1 \|\boldsymbol{A}\|_{2,1} + \frac{t}{2}\left\|\boldsymbol{A} + \boldsymbol{B}_k - \boldsymbol{M}_k + \frac{\boldsymbol{\Lambda}_k}{t}\right\|_F^2, \\ \boldsymbol{B}_{k+1} = \underset{\boldsymbol{B}}{\operatorname{argmin}}\ \lambda_2 \|\boldsymbol{B}\|_* + \frac{t}{2}\left\|\boldsymbol{B} + \boldsymbol{A}_{k+1} - \boldsymbol{M}_k + \frac{\boldsymbol{\Lambda}_k}{t}\right\|_F^2, \\ \boldsymbol{M}_{k+1} = \underset{\boldsymbol{M}}{\operatorname{argmin}} \sum_{i=1}^{N} \|\boldsymbol{Y}_i - r(\boldsymbol{M}\boldsymbol{X}_i)\|_F^2 + \langle \boldsymbol{\Lambda}_k, \boldsymbol{A}_{k+1} + \boldsymbol{B}_{k+1} - \boldsymbol{M} \rangle + \frac{t}{2}\|\boldsymbol{A}_{k+1} + \boldsymbol{B}_{k+1} - \boldsymbol{M}\|_F^2, \\ \boldsymbol{\Lambda}_{k+1} = \boldsymbol{\Lambda}_k + t(\boldsymbol{A}_{k+1} + \boldsymbol{B}_{k+1} - \boldsymbol{M}_{k+1}). \end{cases}$$

Solving $l_{2,1}$-norm

$$\min_{\boldsymbol{A}} \lambda_1 \|\boldsymbol{A}\|_{2,1} + \frac{t}{2} \left\| \boldsymbol{A} + \boldsymbol{B}_k - \boldsymbol{M}_k + \frac{\boldsymbol{\Lambda}_k}{t} \right\|_F^2$$

Closed Form Update Rule[5]

$$\boldsymbol{A}_{k+1} = \text{prox}_{\frac{\lambda_1}{t}\|\cdot\|_{2,1}}(\boldsymbol{M}_k - \boldsymbol{B}_k - \frac{\boldsymbol{\Lambda}_k}{t}),$$

$$\boldsymbol{C} = \boldsymbol{M}_k - \boldsymbol{B}_k - \frac{\boldsymbol{\Lambda}_k}{t},$$

$$[\boldsymbol{A}_{k+1}]_{:,i} = \begin{cases} \frac{\|[\boldsymbol{C}]_{:,i}\|_2 - \frac{\lambda_1}{t}}{\|[\boldsymbol{C}]_{:,i}\|_2}[\boldsymbol{C}]_{:,i}, & \text{if } \|[\boldsymbol{C}]_{:,i}\|_2 > \frac{\lambda_1}{t}; \\ 0, & \text{otherwise.} \end{cases}$$

[5] G. Liu et al., "*Robust recovery of subspace structures by low-rank representation*", *TPAMI, 2013.*

Solving nuclear norm

$$\min_{\boldsymbol{B}} \lambda_2 \|\boldsymbol{B}\|_* + \frac{t}{2}\left\|\boldsymbol{B} + \boldsymbol{A}_{k+1} - \boldsymbol{M}_k + \frac{\boldsymbol{\Lambda}_k}{t}\right\|_F^2$$

Closed Form Update Rule[6]

$$\boldsymbol{B}_{k+1} = \text{prox}_{\frac{\lambda_2}{t}\|\cdot\|_*}(\boldsymbol{M}_k - \boldsymbol{A}_{k+1} - \frac{\boldsymbol{\Lambda}_k}{t}),$$

$$\boldsymbol{D} = \boldsymbol{M}_k - \boldsymbol{A}_{k+1} - \frac{\boldsymbol{\Lambda}_k}{t},$$

$$\boldsymbol{B}_{k+1} = \boldsymbol{U}\mathcal{D}_{\frac{\lambda_2}{t}}(\boldsymbol{\Sigma})\boldsymbol{V}, \quad \text{where } \mathcal{D}_{\frac{\lambda_2}{t}}(\boldsymbol{\Sigma}) = \text{diag}(\{(\sigma_i - \frac{\lambda_2}{t})_+\}).$$

[6] J-F. Cai et al., "*A singular value thresholding algorithm for matrix completion*", *SIOPT, 2010.*

Comparison on *CIFAR-10* dataset

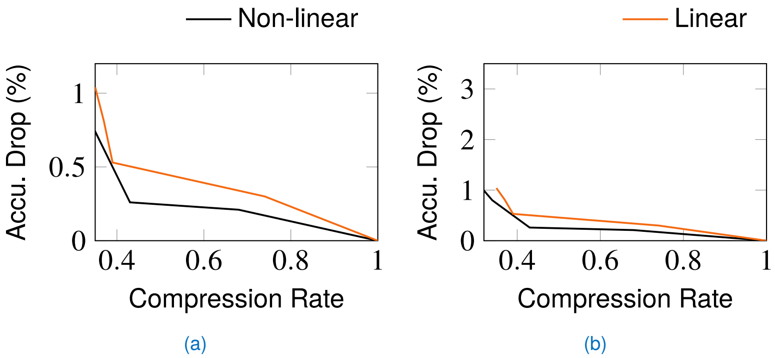
Model	Method	Accuracy ↓	CR	Speed-up
VGG-16	Original	0.00%	1.00	1.00
	ICLR'17[7]	**0.06%**	2.70	1.80
	Ours	0.40%	**4.44**	**2.20**
NIN	Original	0.00%	1.00	1.00
	ICLR'16[8]	1.43%	1.54	1.50
	IJCAI'18[9]	1.43%	1.45	-
	Ours	**0.41%**	**2.77**	**1.70**

[7] Hao Li et al. (2017). "Pruning filters for efficient convnets". In: *Proc. ICLR*.

[8] Cheng Tai et al. (2016). "Convolutional neural networks with low-rank regularization". In: *Proc. ICLR*.

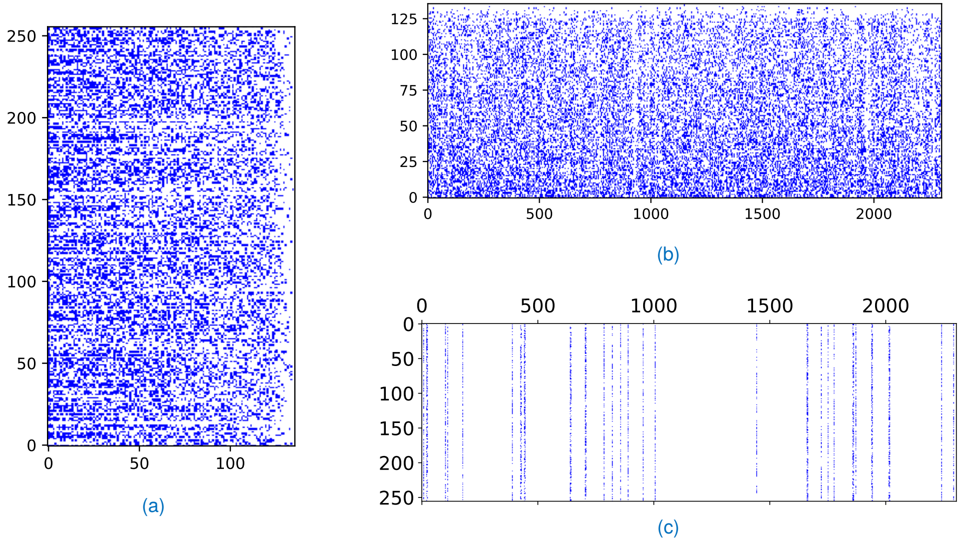
[9] Shiva Prasad Kasiviswanathan, Nina Narodytska, and Hongxia Jin (2018). "Network Approximation using Tensor Sketching". In: *Proc. IJCAI*, pp. 2319–2325.

Preliminary Results



Comparison of reconstructing linear response and non-linear response: (a) layer `conv2-1`; (b) layer `conv3-1`.

Approximation Example



Approximated filters of `conv3-1`. Blue dots have non-zero values. Low-rank filter $\boldsymbol{B}$ with rank 136 is decomposed into $\boldsymbol{UV}$, both of which have rank 136. (a) Matrix $\boldsymbol{U}$; (b) Matrix $\boldsymbol{V}$. (c) Column-wise sparse filter $\boldsymbol{A}$.

Comparison on *ImageNet* dataset

Model	Method	Top-5 Accu.↓	CR	Speed-up
AlexNet	Original	0.00%	1.00	1.00
	ICLR'16[10]	**0.37%**	5.00	**1.82**
	ICLR'16[11]	1.70%	5.46	1.81
	CVPR'18[12]	1.43%	1.50	-
	Ours	1.27%	**5.56**	1.10
GoogleNet	Original	0.00%	1.00	1.00
	ICLR'16[12]	0.42%	2.84	1.20
	ICLR'16[13]	0.24%	1.28	1.23
	CVPR'18[14]	0.21%	1.50	-
	Ours	**0.00%**	**2.87**	**1.35**

[10] Cheng Tai et al. (2016). "Convolutional neural networks with low-rank regularization". In: *Proc. ICLR*.

[11] Yong-Deok Kim et al. (2016). "Compression of deep convolutional neural networks for fast and low power mobile applications". In: *Proc. ICLR*.

[12] Ruichi Yu et al. (2018). "NISP: Pruning networks using neuron importance score propagation". In: *Proc. CVPR*, pp. 9194–9203.