



CMSC 5743

Efficient Computing of Deep Neural Networks

Model 02: Low Rank Decomposition

Bei Yu

CSE Department, CUHK

byu@cse.cuhk.edu.hk

(Latest update: February 20, 2026)

2026 Spring



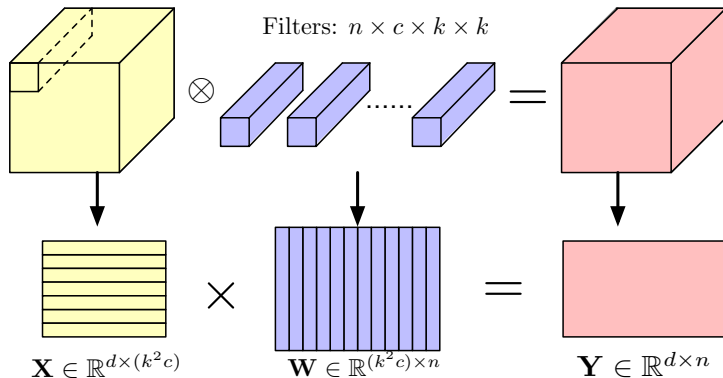
- ① Re-visit DNN Pruning
- ② Singular Value Decomposition (SVD)
- ③ How to Compute SVD?
- ④ Applications to DNN Decomposition
- ⑤ Tensor Decomposition
- ⑥ Matrix Regression Approach



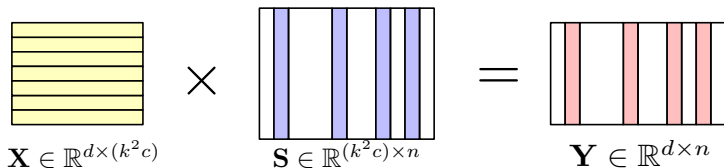
- 1 Re-visit DNN Pruning
- 2 Singular Value Decomposition (SVD)
- 3 How to Compute SVD?
- 4 Applications to DNN Decomposition
- 5 Tensor Decomposition
- 6 Matrix Regression Approach



Re-visit DNN Pruning



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


$$\mathbf{X} \in \mathbb{R}^{d \times (k^2 c)} \quad \times \quad \mathbf{S} \in \mathbb{R}^{(k^2 c) \times n} = \mathbf{Y} \in \mathbb{R}^{d \times n}$$

Sparse DNN

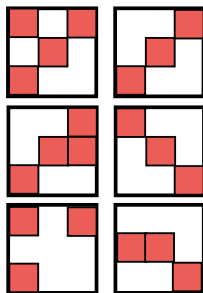
- *Sparsification*: weight pruning;
- *Compression*: compressed sparse format for storage;
- *Potential acceleration*: sparse matrix multiplication algorithm.

Exploring the Granularity of Sparsity that is Hardware-friendly

4 types of pruning granularity



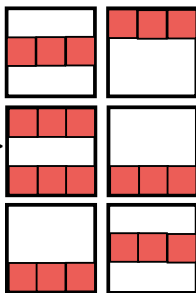
irregular sparsity



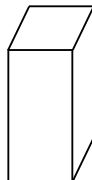
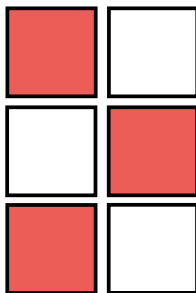
[Han et al, NIPS'15]



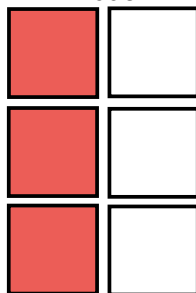
regular sparsity



more regular sparsity



fully-dense model



[Molchanov et al, ICLR'17]



The diagram illustrates the low-rank decomposition of a matrix $\mathbf{X}$ into two matrices $\mathbf{U}$ and $\mathbf{V}$, which are then multiplied to produce the result $\mathbf{Y}$. The matrix $\mathbf{X}$ is represented by a yellow rectangle with 6 horizontal lines. The matrix $\mathbf{U}$ is represented by a purple rectangle with 4 vertical lines. The matrix $\mathbf{V}$ is represented by a purple rectangle with 4 horizontal lines. The result $\mathbf{Y}$ is represented by a red rectangle. The equation is shown as $\mathbf{X} \times \mathbf{U} \times \mathbf{V} = \mathbf{Y}$.

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

Low-rank DNN

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



- 1 Re-visit DNN Pruning
- 2 Singular Value Decomposition (SVD)
- 3 How to Compute SVD?
- 4 Applications to DNN Decomposition
- 5 Tensor Decomposition
- 6 Matrix Regression Approach



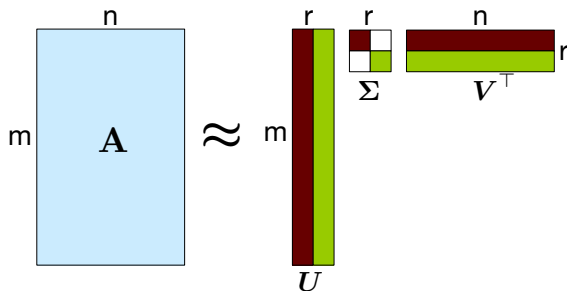
Singular Value Decomposition (SVD)



The diagram illustrates the matrix decomposition $A \sim U \Sigma V^T$. Matrix A is a pink rectangle with dimensions m (height) and n (width). Matrix U is a green rectangle with dimensions m (height) and r (width). Matrix Σ is a purple square with dimensions r (height) and r (width). Matrix V^T is a yellow rectangle with dimensions r (height) and n (width). The matrices are connected by multiplication symbols ($\times$) and an approximation symbol ($\sim$).

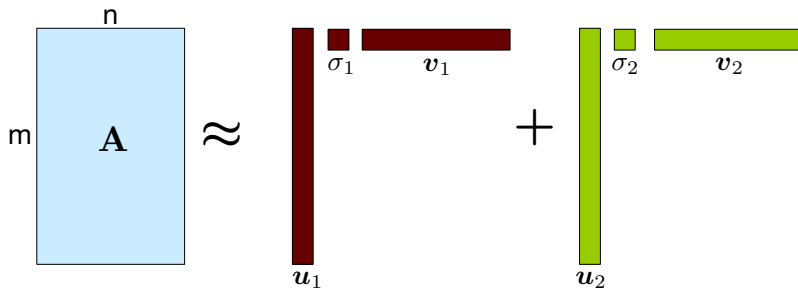
- Gives a decomposition of any matrix into a product of **three matrices**.
- There are strong constraints on the form of each of these matrices
- Results in a **unique** decomposition
- From this decomposition, you can choose **any number r** of intermediate concepts (latent factors)
- In a way that minimizes the **reconstruction error**

$$A \approx U \Sigma V^T = \sum_{i=1}^r \sigma_i u_i \otimes v_i^T$$



- A : input data matrix ($m \times n$ matrix)
- U : left singular vectors ($m \times r$ matrix)
- Σ : Singular values ($r \times r$ diagonal matrix)
- V : right singular vectors ($n \times r$ matrix)

$$A \approx U \Sigma V^T = \sum_{i=1}^r \sigma_i u_i \otimes v_i^T$$



- σ_i : scalar
- u_i, v_i : vector
- If we set $\sigma_2 = 0$, then the green columns may as well not exist.

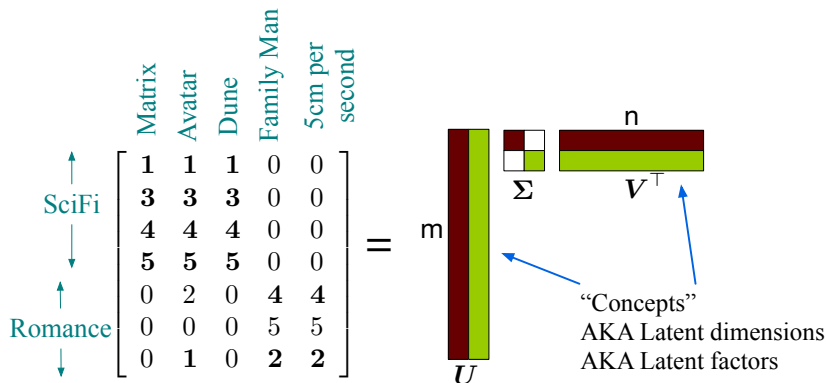


Property

It is **always** possible to decompose a real matrix A into $A = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$

- $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$: **unique**
- $\mathbf{U}, \mathbf{V}$: column **orthonormal**
 - $\mathbf{U}^\top \mathbf{U} = \mathbf{I}; \mathbf{V}^\top \mathbf{V} = \mathbf{I}$ ($\mathbf{I}$: identity matrix)
 - Columns are orthogonal unit vectors
- $\mathbf{\Sigma}$: diagonal
 - Entries (**singular values**) are **non-negative**
 - Sorted in decreasing order ($\sigma_1 \geq \sigma_2 \geq \dots \geq 0$)

Consider a matrix. What does SVD do?



- Ratings matrix where each column corresponds to a movie and each row to a user.
- First 4 users prefer SciFi, while others prefer Romance.

Example: Users-to-Movies



$$\begin{array}{c}
 \uparrow \text{SciFi} \\
 \downarrow \\
 \uparrow \text{Romance} \\
 \downarrow
 \end{array}
 \begin{array}{c}
 \text{Matrix} \\
 \text{Avatar} \\
 \text{Dune} \\
 \text{Family Man} \\
 \text{5cm per second}
 \end{array}
 \begin{bmatrix}
 1 & 1 & 1 & 0 & 0 \\
 3 & 3 & 3 & 0 & 0 \\
 4 & 4 & 4 & 0 & 0 \\
 5 & 5 & 5 & 0 & 0 \\
 0 & 2 & 0 & 4 & 4 \\
 0 & 0 & 0 & 5 & 5 \\
 0 & 1 & 0 & 2 & 2
 \end{bmatrix}
 =
 \begin{bmatrix}
 \mathbf{0.13} & 0.02 & -0.01 \\
 \mathbf{0.41} & 0.07 & -0.03 \\
 \mathbf{0.55} & 0.09 & -0.04 \\
 \mathbf{0.68} & 0.11 & -0.05 \\
 0.15 & \mathbf{-0.59} & \mathbf{0.65} \\
 0.07 & \mathbf{-0.73} & \mathbf{-0.67} \\
 0.07 & \mathbf{-0.29} & \mathbf{0.32}
 \end{bmatrix}
 \times
 \begin{bmatrix}
 \mathbf{12.4} & 0 & 0 \\
 0 & \mathbf{9.5} & 0 \\
 0 & 0 & \mathbf{1.3}
 \end{bmatrix}$$

$$\times
 \begin{bmatrix}
 \mathbf{0.56} & \mathbf{0.59} & \mathbf{0.56} & 0.09 & 0.09 \\
 0.12 & -0.02 & 0.12 & \mathbf{-0.69} & \mathbf{-0.69} \\
 0.40 & \mathbf{-0.80} & 0.40 & 0.09 & 0.09
 \end{bmatrix}$$

- U : “user-to-concept” factor matrix
- V : “movie-to-concept” factor matrix
- Σ : its diagonal elements: “strength” of each concept

Example: Users-to-Movies



$$\begin{array}{c} \uparrow \\ \text{SciFi} \\ \downarrow \\ \uparrow \\ \text{Romance} \\ \downarrow \end{array} \begin{array}{c} \text{Matrix} \\ \text{Avatar} \\ \text{Dune} \\ \text{Family Man} \\ \text{5cm per} \\ \text{second} \end{array} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} = \begin{array}{c} \text{SciFi-concept} \\ \text{Romance-concept} \end{array} \begin{bmatrix} \textcircled{0.13} & \textcircled{0.02} & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & 0.32 \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix} \\
 \times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

- U : “user-to-concept” factor matrix
- V : “movie-to-concept” factor matrix
- Σ : its diagonal elements: “strength” of each concept

Example: Users-to-Movies



$$\begin{array}{c} \uparrow \\ \text{SciFi} \\ \downarrow \\ \uparrow \\ \text{Romance} \\ \downarrow \end{array} \begin{array}{c} \text{Matrix} \\ \text{Avatar} \\ \text{Dune} \\ \text{Family Man} \\ \text{5cm per} \\ \text{second} \end{array} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & 0.32 \end{bmatrix} \times \begin{array}{c} \text{"strength" of} \\ \text{SciFi-concept} \\ \downarrow \end{array} \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix}$$

$$\times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

- U : "user-to-concept" factor matrix
- V : "movie-to-concept" factor matrix
- Σ : its diagonal elements: "strength" of each concept

Example: Users-to-Movies

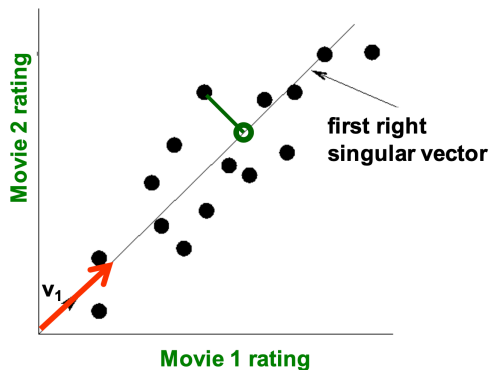


$$\begin{array}{c}
 \uparrow \text{SciFi} \\
 \downarrow \\
 \uparrow \text{Romance} \\
 \downarrow
 \end{array}
 \begin{bmatrix}
 \text{Matrix} & \text{Avatar} & \text{Dune} & \text{Family Man} & \text{5cm per second} \\
 1 & 1 & 1 & 0 & 0 \\
 3 & 3 & 3 & 0 & 0 \\
 4 & 4 & 4 & 0 & 0 \\
 5 & 5 & 5 & 0 & 0 \\
 0 & 2 & 0 & 4 & 4 \\
 0 & 0 & 0 & 5 & 5 \\
 0 & 1 & 0 & 2 & 2
 \end{bmatrix}
 =
 \begin{bmatrix}
 0.13 & 0.02 & -0.01 \\
 0.41 & 0.07 & -0.03 \\
 0.55 & 0.09 & -0.04 \\
 0.68 & 0.11 & -0.05 \\
 0.15 & -0.59 & 0.65 \\
 0.07 & -0.73 & -0.67 \\
 0.07 & -0.29 & 0.32
 \end{bmatrix}
 \times
 \begin{bmatrix}
 12.4 & 0 & 0 \\
 0 & 9.5 & 0 \\
 0 & 0 & 1.3
 \end{bmatrix}$$

$$\times
 \begin{bmatrix}
 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\
 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\
 0.40 & -0.80 & 0.40 & 0.09 & 0.09
 \end{bmatrix}$$

Note: In the original image, blue arrows point from the SciFi/Romance labels to the first and last rows of the matrix. Purple arrows point from the diagonal elements 0.56 and 0.59 to the first and last rows of the matrix. These two values are circled in blue and purple respectively.

- U : “user-to-concept” factor matrix
- V : “movie-to-concept” factor matrix
- Σ : its diagonal elements: “strength” of each concept



- Instead of using two coordinates (x, y) to describe points
- Let's use only one coordinate
- Point's position is its location along vector v_1



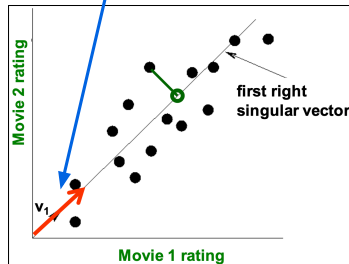
$$\begin{array}{c}
 \begin{array}{c} \uparrow \\ \text{SciFi} \\ \downarrow \\ \uparrow \\ \text{Romance} \\ \downarrow \end{array}
 \begin{array}{c}
 \text{Matrix} \\
 \text{Avatar} \\
 \text{Dune} \\
 \text{Family Man} \\
 \text{5cm per} \\
 \text{second}
 \end{array}
 \begin{bmatrix}
 1 & 1 & 1 & 0 & 0 \\
 3 & 3 & 3 & 0 & 0 \\
 4 & 4 & 4 & 0 & 0 \\
 5 & 5 & 5 & 0 & 0 \\
 0 & 2 & 0 & 4 & 4 \\
 0 & 0 & 0 & 5 & 5 \\
 0 & 1 & 0 & 2 & 2
 \end{bmatrix}
 =
 \begin{bmatrix}
 0.13 & 0.02 & -0.01 \\
 0.41 & 0.07 & -0.03 \\
 0.55 & 0.09 & -0.04 \\
 0.68 & 0.11 & -0.05 \\
 0.15 & -0.59 & 0.65 \\
 0.07 & -0.73 & -0.67 \\
 0.07 & -0.29 & 0.32
 \end{bmatrix}
 \times
 \begin{bmatrix}
 12.4 & 0 & 0 \\
 0 & 9.5 & 0 \\
 0 & 0 & 1.3
 \end{bmatrix}
 \times
 \begin{bmatrix}
 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\
 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\
 0.40 & -0.80 & 0.40 & 0.09 & 0.09
 \end{bmatrix}
 \end{array}$$

SVD: Dimensionality Reduction



$$\begin{array}{c} \text{SciFi} \\ \updownarrow \\ \text{Romance} \end{array} \begin{array}{c} \text{Matrix} \\ \text{Avatar} \\ \text{Dune} \\ \text{Family Man} \\ \text{5cm per second} \end{array} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & 0.32 \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix}$$

$$\times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix}$$



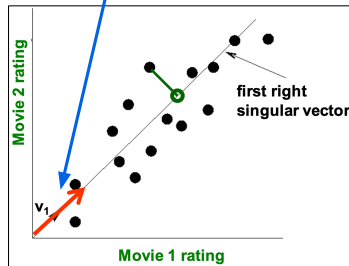
SVD: Dimensionality Reduction

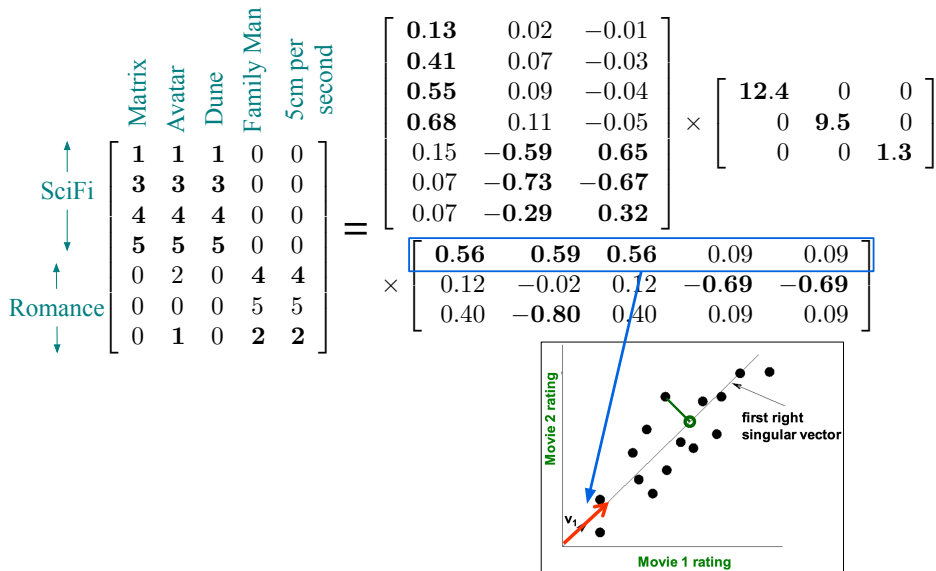


		Matrix	Avatar	Dune	Family Man	5cm per second
		1	1	1	0	0
SciFi	↑	3	3	3	0	0
	↓	4	4	4	0	0
		5	5	5	0	0
	↑	0	2	0	4	4
Romance		0	0	0	5	5
	↓	0	1	0	2	2

$$= \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & 0.32 \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix}$$

$$\times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix}$$





Q: How to coordinates of the points in the projection axis?



$$\begin{array}{c}
 \uparrow \text{SciFi} \\
 \downarrow \\
 \uparrow \text{Romance} \\
 \downarrow
 \end{array}
 \begin{array}{c}
 \text{Matrix} \\
 \text{Avatar} \\
 \text{Dune} \\
 \text{Family Man} \\
 \text{5 cm per} \\
 \text{second}
 \end{array}
 \begin{bmatrix}
 1 & 1 & 1 & 0 & 0 \\
 3 & 3 & 3 & 0 & 0 \\
 4 & 4 & 4 & 0 & 0 \\
 5 & 5 & 5 & 0 & 0 \\
 0 & 2 & 0 & 4 & 4 \\
 0 & 0 & 0 & 5 & 5 \\
 0 & 1 & 0 & 2 & 2
 \end{bmatrix}
 =
 \begin{bmatrix}
 0.13 & 0.02 & -0.01 \\
 0.41 & 0.07 & -0.03 \\
 0.55 & 0.09 & -0.04 \\
 0.68 & 0.11 & -0.05 \\
 0.15 & -0.59 & 0.65 \\
 0.07 & -0.73 & -0.67 \\
 0.07 & -0.29 & 0.32
 \end{bmatrix}
 \times
 \begin{bmatrix}
 12.4 & 0 & 0 \\
 0 & 9.5 & 0 \\
 0 & 0 & 1.3
 \end{bmatrix}$$

$$\times
 \begin{bmatrix}
 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\
 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\
 0.40 & -0.80 & 0.40 & 0.09 & 0.09
 \end{bmatrix}$$

- Set **small** singular values to **zero**
- The above example is rank-2 approximation
- We could also do rank-1 approximation.
- The larger the rank, the more accurate the approximation

SVD: Another Interpretation



$$\begin{array}{c}
 \begin{array}{c} \uparrow \\ \text{SciFi} \\ \downarrow \\ \uparrow \\ \text{Romance} \\ \downarrow \end{array}
 \begin{array}{c} \text{Matrix} \\ \text{Avatar} \\ \text{Dune} \\ \text{Family Man} \\ \text{5 cm per second} \end{array}
 \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix}
 \approx
 \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & 0.32 \end{bmatrix}
 \times
 \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.8 \end{bmatrix}
 \end{array}$$

$$\times
 \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.69 & 0.09 \end{bmatrix}$$

- Set **small** singular values to **zero**
- The above example is rank-2 approximation
- We could also do rank-1 approximation.
- The larger the rank, the more accurate the approximation



- Reconstruction error is quantified by the **Frobenius** norm:

$$||\mathbf{M}||_F = \sqrt{\sum_{ij} M_{ij}^2}$$

$$||\mathbf{A} - \mathbf{B}||_F = \sqrt{\sum_{ij} (A_{ij} - B_{ij})^2}$$



- Reconstruction error is quantified by the **Frobenius** norm:

$$\|\mathbf{M}\|_F = \sqrt{\sum_{ij} M_{ij}^2}$$
$$\|\mathbf{A} - \mathbf{B}\|_F = \sqrt{\sum_{ij} (A_{ij} - B_{ij})^2}$$

Theorem:

The **best** rank-1 approximation to $\mathbf{A}$ is $\sigma_1 \mathbf{u}_1 \mathbf{v}_1^\top$, where σ_1 is the largest singular value, $\mathbf{u}_1$ is the first left singular vector, and $\mathbf{v}_1$ is the first right singular vector of $\mathbf{A}$.



- Let the **energy** of a set of singular values be the sum of their **squares**.
- Pick r so the retained singular values have at least **90%** of the total energy



- Let the **energy** of a set of singular values be the sum of their **squares**.
- Pick r so the retained singular values have at least **90%** of the total energy

Example:

- With singular values 12.4, 9.5, and 1.3, total energy = 245.7
- If we drop 1.3, whose square is only 1.7
- We are left with energy 244, or over 99% of the total



Please prove: $AA^{\top}$ and $A^{\top}A$ are with the same eigenvalues.

Proof:



- 1 Re-visit DNN Pruning
- 2 Singular Value Decomposition (SVD)
- 3 How to Compute SVD?
- 4 Applications to DNN Decomposition
- 5 Tensor Decomposition
- 6 Matrix Regression Approach



How to Compute SVD?



- First we need finding the **principal eigenvalue** (the largest one)
- and the corresponding **eigenvector** of a symmetric matrix
- Note: M is **symmetric** if $M_{ij} = M_{ji}$ for all i and j

Method¹:

- 1 Start with any “guess eigenvecor” x_0
- 2 Construct $x_{k+1} = \frac{Mx_k}{||Mx_k||}$, for $k = 0, 1, \dots$ ²
- 3 Stop when consecutive x_i show little change

¹Power iteration: https://en.wikipedia.org/wiki/Power_iteration

² $||\cdot||$ denotes the Frobenius norm



$$M = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}, \quad x_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$



$$M = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}, \quad x_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\frac{Mx_0}{\|Mx_0\|} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} / \sqrt{34} = \begin{bmatrix} 0.51 \\ 0.86 \end{bmatrix} = x_1$$



$$M = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}, \quad x_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\frac{Mx_0}{\|Mx_0\|} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} / \sqrt{34} = \begin{bmatrix} 0.51 \\ 0.86 \end{bmatrix} = x_1$$

$$\frac{Mx_1}{\|Mx_1\|} = \begin{bmatrix} 2.23 \\ 3.60 \end{bmatrix} / \sqrt{17.93} = \begin{bmatrix} 0.53 \\ 0.85 \end{bmatrix} = x_2$$



With principal eigenvector x , we can find eigenvalue λ by

$$\lambda = x^{\top} M x$$



With principal eigenvector $\mathbf{x}$, we can find eigenvalue λ by

$$\lambda = \mathbf{x}^\top \mathbf{M} \mathbf{x}$$

Prove:

- ① We know $\mathbf{x} \cdot \lambda = \mathbf{M} \mathbf{x}$ if λ is the eigenvalue
- ② Multiply both sides by $\mathbf{x}^\top$ on the left
- ③ Since $\mathbf{x}^\top \mathbf{x} = 1$, we have $\lambda = \mathbf{x}^\top \mathbf{M} \mathbf{x}$



With principal eigenvector $\mathbf{x}$, we can find eigenvalue λ by

$$\lambda = \mathbf{x}^\top \mathbf{M} \mathbf{x}$$

Prove:

- ① We know $\mathbf{x} \cdot \lambda = \mathbf{M} \mathbf{x}$ if λ is the eigenvalue
- ② Multiply both sides by $\mathbf{x}^\top$ on the left
- ③ Since $\mathbf{x}^\top \mathbf{x} = 1$, we have $\lambda = \mathbf{x}^\top \mathbf{M} \mathbf{x}$

Example: if we take $\mathbf{x}^\top = [0.53, 0.85]$, then

$$\lambda = [0.53, 0.85] \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 0.53 \\ 0.85 \end{bmatrix} = 4.25$$



- 1 Eliminate the portion of the matrix M that can be generated by the first eigenpair, λ and x :

$$M_* = M - \lambda x x^\top$$

- 2 Recursively find the principal eigenpair for M_*
- 3 Eliminate the effect of that pair, and so on



- 1 Eliminate the portion of the matrix M that can be generated by the first eigenpair, λ and x :

$$M_* = M - \lambda x x^\top$$

- 2 Recursively find the principal eigenpair for M_*
- 3 Eliminate the effect of that pair, and so on

Example:

$$M_* = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} - 4.25 \begin{bmatrix} 0.53 \\ 0.85 \end{bmatrix} [0.53, 0.85] = \begin{bmatrix} -0.19 & 0.09 \\ 0.09 & 0.07 \end{bmatrix}$$



- 1 Start by supposing $A = U\Sigma V^T$



- ① Start by supposing $A = U\Sigma V^T$
- ② $A^T = (U\Sigma V^T)^T = (V^T)^T \Sigma^T U^T = V\Sigma U^T$



- ① Start by supposing $A = U\Sigma V^\top$
- ② $A^\top = (U\Sigma V^\top)^\top = (V^\top)^\top \Sigma^\top U^\top = V\Sigma U^\top$
- ③ $A^\top A = V\Sigma U^\top U\Sigma V^\top = V\Sigma^2 V^\top$
 - Note: U is orthonormal, so $U^\top U$ is an identity matrix
 - Note: Σ^2 is a diagonal matrix whose i -th element is the square of the i -th element of Σ



- ① Start by supposing $A = U\Sigma V^\top$
- ② $A^\top = (U\Sigma V^\top)^\top = (V^\top)^\top \Sigma^\top U^\top = V\Sigma U^\top$
- ③ $A^\top A = V\Sigma U^\top U\Sigma V^\top = V\Sigma^2 V^\top$
 - Note: U is orthonormal, so $U^\top U$ is an identity matrix
 - Note: Σ^2 is a diagonal matrix whose i -th element is the square of the i -th element of Σ
- ④ $A^\top AV = V\Sigma^2 V^\top V = V\Sigma^2$
 - Note: V is also orthonormal



- ① Start by supposing $A = U\Sigma V^\top$
- ② $A^\top = (U\Sigma V^\top)^\top = (V^\top)^\top \Sigma^\top U^\top = V\Sigma U^\top$
- ③ $A^\top A = V\Sigma U^\top U\Sigma V^\top = V\Sigma^2 V^\top$
 - Note: U is orthonormal, so $U^\top U$ is an identity matrix
 - Note: Σ^2 is a diagonal matrix whose i-th element is the square of the i-th element of Σ
- ④ $A^\top AV = V\Sigma^2 V^\top V = V\Sigma^2$
 - Note: V is also orthonormal
- ⑤ We can find V & Σ by finding the eigenpairs for $A^\top A$



- ① Start by supposing $A = U\Sigma V^\top$
- ② $A^\top = (U\Sigma V^\top)^\top = (V^\top)^\top \Sigma^\top U^\top = V\Sigma U^\top$
- ③ $A^\top A = V\Sigma U^\top U\Sigma V^\top = V\Sigma^2 V^\top$
 - Note: U is orthonormal, so $U^\top U$ is an identity matrix
 - Note: Σ^2 is a diagonal matrix whose i-th element is the square of the i-th element of Σ
- ④ $A^\top A V = V\Sigma^2 V^\top V = V\Sigma^2$
 - Note: V is also orthonormal
- ⑤ We can find V & Σ by finding the eigenpairs for $A^\top A$
- ⑥ Symmetric argument: $AA^\top$ gives us U



- ① Start by supposing $A = U\Sigma V^\top$
- ② $A^\top = (U\Sigma V^\top)^\top = (V^\top)^\top \Sigma^\top U^\top = V\Sigma U^\top$
- ③ $A^\top A = V\Sigma U^\top U\Sigma V^\top = V\Sigma^2 V^\top$
 - Note: U is orthonormal, so $U^\top U$ is an identity matrix
 - Note: Σ^2 is a diagonal matrix whose i -th element is the square of the i -th element of Σ
- ④ $A^\top AV = V\Sigma^2 V^\top V = V\Sigma^2$
 - Note: V is also orthonormal
- ⑤ We can find V & Σ by finding the eigenpairs for $A^\top A$
- ⑥ Symmetric argument: $AA^\top$ gives us U



Example:

Calculate SVD of matrix $A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}$.



Example:

Calculate SVD of matrix $A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}$.

Solution:

$$A^T A = \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}$$



Example:

Calculate SVD of matrix $A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}$.

Solution:

$$A^T A = \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}$$

$$v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$



Example:

Calculate SVD of matrix $A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}$.

Solution:

$$A^T A = \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}$$

$$v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$

$$\rightarrow \lambda_1 = v_1^T A^T A v_1 = 10; \text{ therefore } \sigma_1 = \sqrt{\lambda_1} = \sqrt{10}$$



Example:

Calculate SVD of matrix $A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}$.

Solution:

$$A^T A = \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}$$

$$v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$

$$\rightarrow \lambda_1 = v_1^T A^T A v_1 = 10; \text{ therefore } \sigma_1 = \sqrt{\lambda_1} = \sqrt{10}$$

$$\rightarrow \lambda_2 = v_2^T A^T A v_2 = 0; \text{ therefore } \sigma_2 = \sqrt{\lambda_2} = 0$$



Example:

Calculate SVD of matrix $A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}$.

Solution:

$$A^T A = \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}$$

$$v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$

$$\rightarrow \lambda_1 = v_1^T A^T A v_1 = 10; \text{ therefore } \sigma_1 = \sqrt{\lambda_1} = \sqrt{10}$$

$$\rightarrow \lambda_2 = v_2^T A^T A v_2 = 0; \text{ therefore } \sigma_2 = \sqrt{\lambda_2} = 0$$

...

$$A = U \Sigma V^T = \begin{bmatrix} 1/\sqrt{5} & -2/\sqrt{5} & 0 \\ 2/\sqrt{5} & 1/\sqrt{5} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{10} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$



To compute the full SVD using specialized methods:

- $\mathcal{O}(nm^2)$ or $\mathcal{O}(n^2m)$ (whichever is less)

But:

- Less work, if we just want singular values
- Or if we want the **first** k singular vectors
- Or if the matrix is sparse

Implemented in linear algebra packages link

- LINPACK, Matlab, SPlus, Mathematica, ...



- 1 Re-visit DNN Pruning
- 2 Singular Value Decomposition (SVD)
- 3 How to Compute SVD?
- 4 Applications to DNN Decomposition**
- 5 Tensor Decomposition
- 6 Matrix Regression Approach



Applications to DNN Decomposition



This CVPR2015 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

Efficient and Accurate Approximations of Nonlinear Convolutional Networks

Xiangyu Zhang^{1*} Jianhua Zou¹
¹Xi'an Jiaotong University

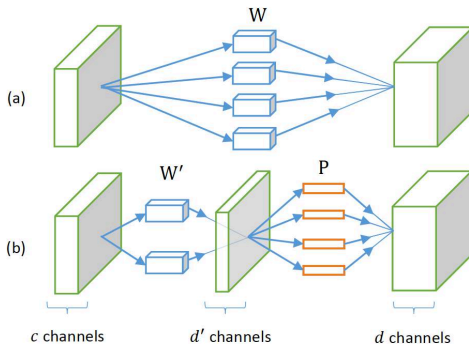
Xiang Ming^{1*}

Kaiming He² Jian Sun²
²Microsoft Research

³Xiangyu Zhang et al. (2015). “Efficient and accurate approximations of nonlinear convolutional networks”. In: *Proc. CVPR*, pp. 1984–1992.

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 ($1 \times 1 \times d'$)





Low Rank Approximation for Conv

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



Published as a conference paper at ICLR 2016

CONVOLUTIONAL NEURAL NETWORKS WITH LOW-RANK REGULARIZATION

Cheng Tai¹, Tong Xiao², Yi Zhang³, Xiaogang Wang², Weinan E¹

¹The Program in Applied and Computational Mathematics, Princeton University

²Department of Electronic Engineering, The Chinese University of Hong Kong

³Department 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

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

⁴Cheng Tai et al. (2016). “Convolutional neural networks with low-rank regularization”. In: *Proc. ICLR*.



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 $\mathcal{W}$

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

where K is a hyper-parameter controlling the rank, $\mathcal{H} \in \mathbb{R}^{N \times 1 \times d \times R}$ 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

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



Complexity Analysis

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

Approximate Parameters H and V

- Minimizing the objective function

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

- 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

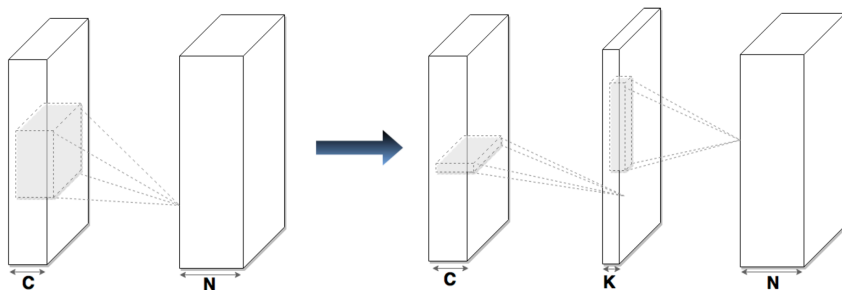
$$j_1 = (i_1 - 1)d + i_2, \quad j_2 = (i_4 - 1)d + i_3$$

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

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

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

- The proposed parametrization for low-rank regularization.



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



Training Low-rank Constrained CNN From Scratch

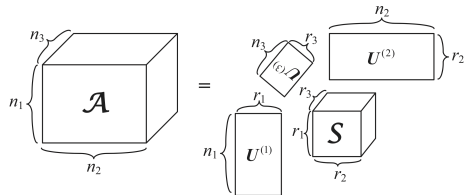
- The effect of SVD Decomposition
Each convolutional layer is parameterized as the composition of two convolutional layers,
- Exploding and vanishing gradients especially for large networks
- Batch Normalization can handle this problem
(Recall the theory of Batch Normalization)



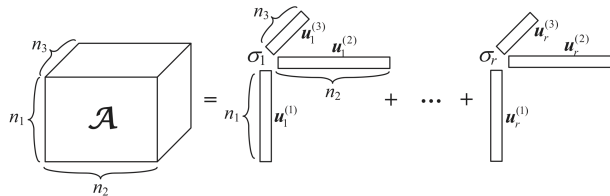
- 1 Re-visit DNN Pruning
- 2 Singular Value Decomposition (SVD)
- 3 How to Compute SVD?
- 4 Applications to DNN Decomposition
- 5 Tensor Decomposition**
- 6 Matrix Regression Approach



Tensor Decomposition



(a) Tucker-decomposition



(b) CP-decomposition



Published as a conference paper at ICLR 2016

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

Yong-Deok Kim¹, Eunhyeok Park², Sungjoo Yoo², Taelim Choi¹, Lu Yang¹ & Dongjun Shin¹

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

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

- 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.

⁵Yong-Deok Kim et al. (2016). “Compression of deep convolutional neural networks for fast and low power mobile applications”. In: *Proc. ICLR*.

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 $\mathcal{K}$ has the form:

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

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

- 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:

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

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:

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

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

- Tucker-2 decompositions for speeding-up a convolution

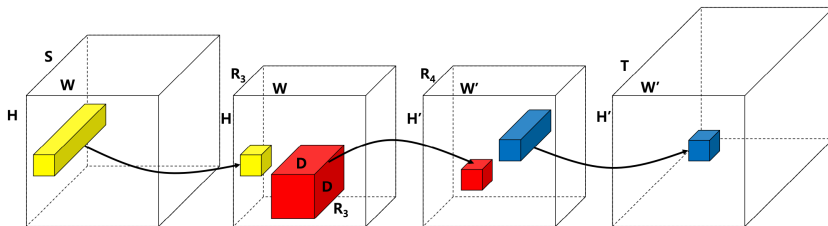


Figure 3: Tucker-2 decompositions for speeding-up a convolution. Each transparent box corresponds to 3-way tensor $\mathcal{X}$, $\mathcal{Z}$, $\mathcal{Z}'$, and $\mathcal{Y}$ in (3-5), with two frontal sides corresponding to spatial dimensions. Arrows represent linear mappings and illustrate how scalar values on the right are computed. Yellow tube, red box, and blue tube correspond to 1×1 , $D \times D$, and 1×1 convolution in (3), (4), and (5) respectively.

- 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

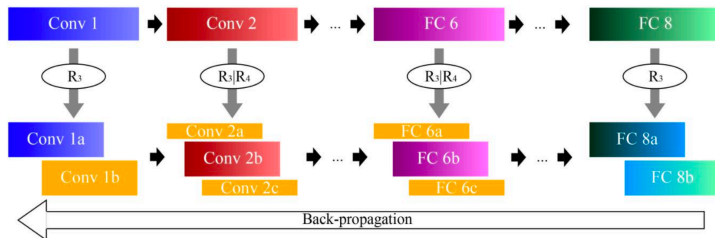


Rank Selection With Global Analytic VBMF

- Motivation: The rank- $(R_3; R_4)$ control the trade-off between performance (memory, speed, energy) improvement and accuracy loss.
- Method: variational Bayesian atrix factorization⁶
- Advantages: VBMF can automatically find noise variance, rank and even provide theoretical condition for perfect rank recovery

⁶Shinichi 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.

- 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.



Published as a conference paper at ICLR 2015

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

Vadim Lebedev^{1,2}, Yaroslav Ganin¹, Maksim Rakhuba^{1,3}, Ivan Oseledets^{1,4}, and Victor Lempitsky¹

¹Skolkovo Institute of Science and Technology (Skoltech), Moscow, Russia

²Yandex, Moscow, Russia

³Moscow Institute of Physics and Technology, Moscow Region, Russia

⁴Institute of Numerical Mathematics RAS, Moscow, Russia

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

⁷Vadim Lebedev et al. (2015). “Speeding-up convolutional neural networks using fine-tuned CP-decomposition”. In: *Proc. ICLR*.

Advantages

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

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 $n_1 \times \dots \times n_d$ a CP-decomposition has the following form

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

where the minimal possible R is called canonical rank.

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

Notes:

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



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

- Substitute 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

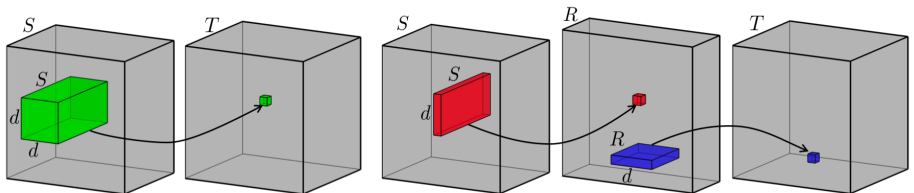
$$U^s(i, j, r) = \sum_{s=1}^S K^s(s, r) U(i, j, s)$$

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

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

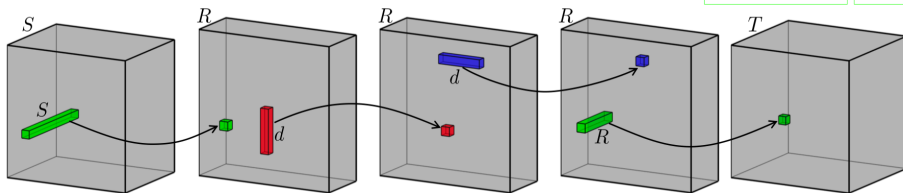
$$V(x, y, t) = \sum_{r=1}^R K^t(t, r) U^{syx}(x, y, r)$$

Complexity Comparison:



(a) Full convolution

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



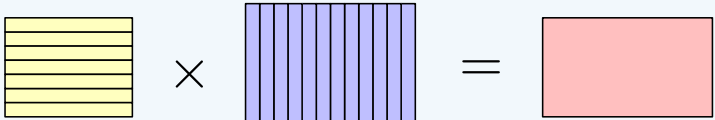
(c) CP-decomposition



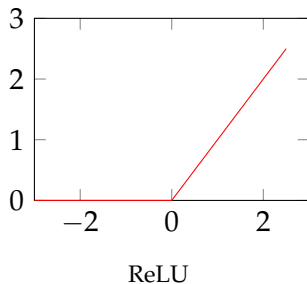
- 1 Re-visit DNN Pruning
- 2 Singular Value Decomposition (SVD)
- 3 How to Compute SVD?
- 4 Applications to DNN Decomposition
- 5 Tensor Decomposition
- 6 Matrix Regression Approach



Matrix Regression Approach


$$\mathbf{X} \in \mathbb{R}^{d \times (k^2 c)} \quad \times \quad \mathbf{W} \in \mathbb{R}^{(k^2 c) \times n} \quad = \quad \mathbf{Y} \in \mathbb{R}^{d \times n}$$

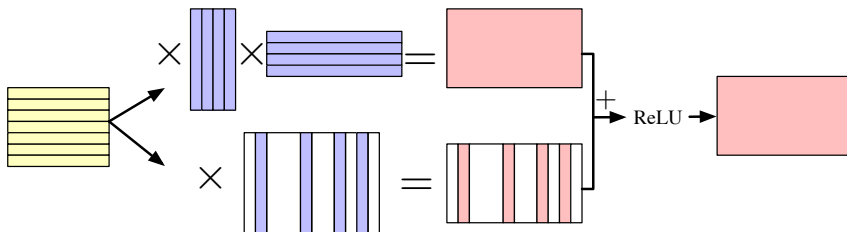
- Matrix approximation: $\mathbf{W} \approx \mathbf{W}'$
- Matrix regression: $\mathbf{Y} = \mathbf{W} \cdot \mathbf{X} \approx \mathbf{W}' \cdot \mathbf{X}$



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

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

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



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

⁸Yuzhe Ma et al. (2019). “A Unified Approximation Framework for Non-Linear Deep Neural Networks”. In: *Proc. ICTAI*.



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.

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

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

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



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

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





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

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

Then the augmented Lagrangian function is

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



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

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



$$\min_A \lambda_1 \|A\|_{2,1} + \frac{t}{2} \left\| A + B_k - M_k + \frac{\Lambda_k}{t} \right\|_F^2$$

Closed Form Update Rule⁹

$$\begin{aligned} A_{k+1} &= \text{prox}_{\frac{\lambda_1}{t} \|\cdot\|_{2,1}} \left(M_k - B_k - \frac{\Lambda_k}{t} \right), \\ C &= M_k - B_k - \frac{\Lambda_k}{t}, \\ [A_{k+1}]_{:,i} &= \begin{cases} \frac{\|[C]_{:,i}\|_2 - \frac{\lambda_1}{t}}{\|[C]_{:,i}\|_2} [C]_{:,i}, & \text{if } \|[C]_{:,i}\|_2 > \frac{\lambda_1}{t}; \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

⁹G. Liu et al., "Robust recovery of subspace structures by low-rank representation", TPAMI, 2013.



$$\min_B \lambda_2 \|B\|_* + \frac{t}{2} \left\| B + A_{k+1} - M_k + \frac{\Lambda_k}{t} \right\|_F^2$$

Closed Form Update Rule¹⁰

$$B_{k+1} = \text{prox}_{\frac{\lambda_2}{t} \|\cdot\|_*} \left(M_k - A_{k+1} - \frac{\Lambda_k}{t} \right),$$

$$D = M_k - A_{k+1} - \frac{\Lambda_k}{t},$$

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

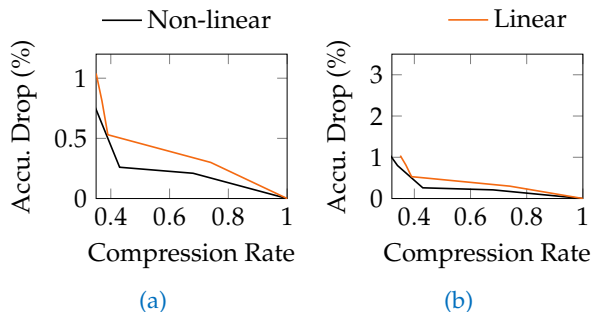
¹⁰J-F. Cai et al., "A singular value thresholding algorithm for matrix completion", SIOPT, 2010.

Model	Method	Accuracy ↓	CR	Speed-up
VGG-16	Original	0.00%	1.00	1.00
	ICLR'17 ¹¹	0.06%	2.70	1.80
	Ours	0.40%	4.44	2.20
NIN	Original	0.00%	1.00	1.00
	ICLR'16 ¹²	1.43%	1.54	1.50
	IJCAI'18 ¹³	1.43%	1.45	-
	Ours	0.41%	2.77	1.70

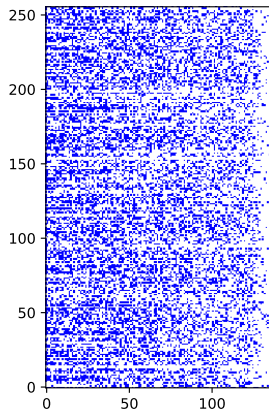
¹¹Hao Li et al. (2017). “Pruning filters for efficient convnets”. In: *Proc. ICLR*.

¹²Cheng Tai et al. (2016). “Convolutional neural networks with low-rank regularization”. In: *Proc. ICLR*.

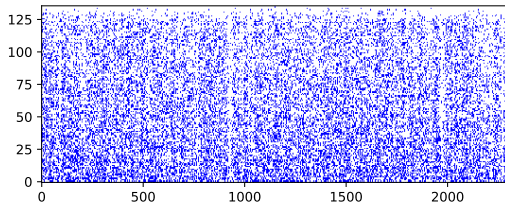
¹³Shiva Prasad Kasiviswanathan, Nina Narodytska, and Hongxia Jin (2018). “Network Approximation using Tensor Sketching”. In: *Proc. IJCAI*, pp. 2319–2325.



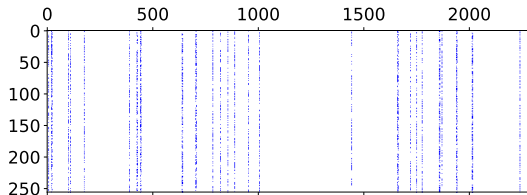
Comparison of reconstructing linear response and non-linear response: (a) layer conv2-1; (b) layer conv3-1.



(a)



(b)



(c)

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

Model	Method	Top-5 Accu.↓	CR	Speed-up
AlexNet	Original	0.00%	1.00	1.00
	ICLR'16 ¹⁴	0.37%	5.00	1.82
	ICLR'16 ¹⁵	1.70%	5.46	1.81
	CVPR'18 ¹⁶	1.43%	1.50	-
	Ours	1.27%	5.56	1.10
GoogleNet	Original	0.00%	1.00	1.00
	ICLR'16 ¹⁰	0.42%	2.84	1.20
	ICLR'16 ¹¹	0.24%	1.28	1.23
	CVPR'18 ¹²	0.21%	1.50	-
	Ours	0.00%	2.87	1.35

¹⁴Cheng Tai et al. (2016). “Convolutional neural networks with low-rank regularization”. In: *Proc. ICLR*.

¹⁵Yong-Deok Kim et al. (2016). “Compression of deep convolutional neural networks for fast and low power mobile applications”. In: *Proc. ICLR*.

¹⁶Ruichi Yu et al. (2018). “NISP: Pruning networks using neuron importance score propagation”. In: *Proc. CVPR*, pp. 9194–9203.