Tensorising Neural Networks

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, Dmitry Vetrov

UBC MLRG 2020 Winter Term 1

Presented by Gursimran Singh (simar)

*Slides adapted from various sources in references

Why tensors?

Many objects in machine learning can be treated as tensors:

- Data cubes (RGB images, videos, different shapes/orientations)
- Weight matrices can be treated as tensors, both in Conv-layers and fully-connected layers

Using tensor decompositions we can compress data!

Motivation

Problem: Neural network is too large to fit into memory.

Approaches:

- Distributed neural network
  - distribute parameters
  - challenge: training [Elastic Averaging SGD (NIPS'15)]

- Model compression
  - reduce required space

Problem Formulation

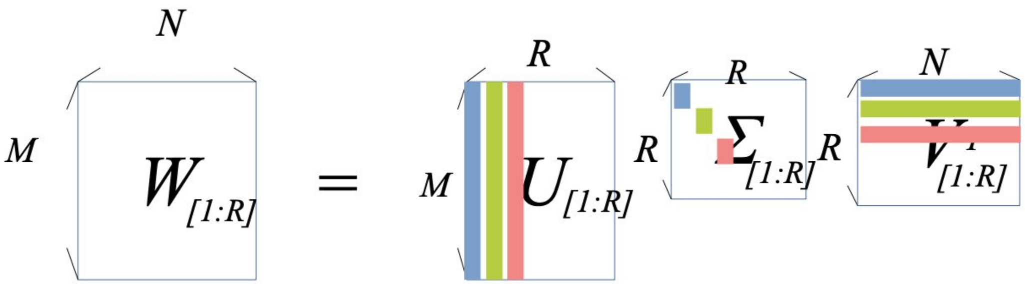
- Given $M \times N$ weight matrix W of a fully-connected layer

$$\boxed{o(x;\theta) = f(W^T x + b)}$$

- Goal
  - reduce space complexity

- Requirement
  - compact with back-propagation

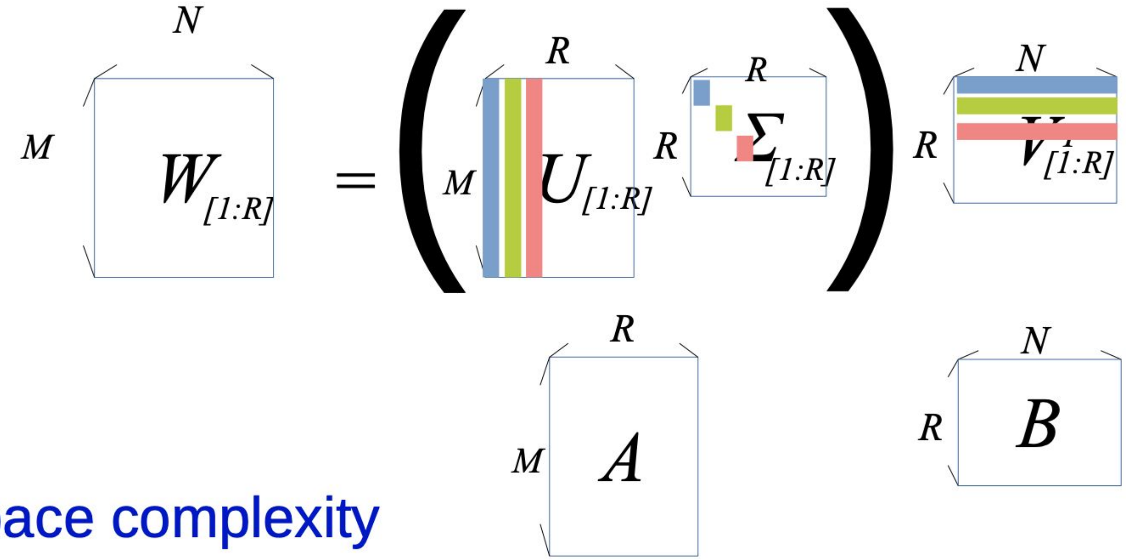
Naive Method: Low-rank SVD



SVD Decomposition

$$W(i,j) = \sum_{r=1}^{R} U(i,r)\Sigma(r,r)V(j,r)^T$$

Naive Method: Low-rank SVD



$$W = \sum_{r=1}^{R} A_r {B_r}^T$$

space complexity

$$\boxed{O(M \times N) \rightarrow O(R(M+N))}$$

Naive Method: Low-rank SVD

By low-rank SVD $$W = \sum_{r=1}^{R} A_r {B_r}^T$$

Instead of updating *W*, $$\frac{\partial E}{\partial W}$$

update components

$$\frac{\partial E}{\partial A_r} \qquad \frac{\partial E}{\partial B_r}$$

Naive Method: Low-rank SVD

- To integrated with back-propagation

$$o = f(W^T x + b)$$

  - have calculate 3 gradients:

$$W = \sum_{r=1}^{R} A_r B_r{}^T$$

    - objective wrt parameter $\frac{\partial E}{\partial o}$
    - output wrt input $\frac{\partial o}{\partial x}$

No change in these

    - output wrt parameter

$$\frac{\partial o}{\partial A_r} = f'(W^T x + b)\, B_r \mathbf{1}^T x$$

$$\frac{\partial o}{\partial B_r} = f'(W^T x + b)\, \mathbf{1} A_r{}^T x$$

Simple equations to compute gradients

TT-Decomposition: Two ideas to do better

Low-rank SVD works but can we do better than that?

Two key Ideas:

recursively applying low-rank SVD

1)
$$r(M+N) \leq MN$$

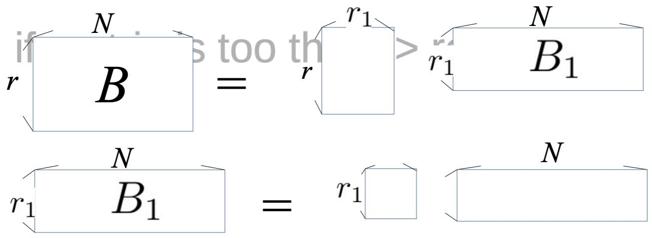
Two ideas to do better

Low-rank SVD works but can we do better than that?

Two key Ideas:

recursively applying low-rank SVD

1) $$r(M+N) \leq MN$$



Two ideas to do better

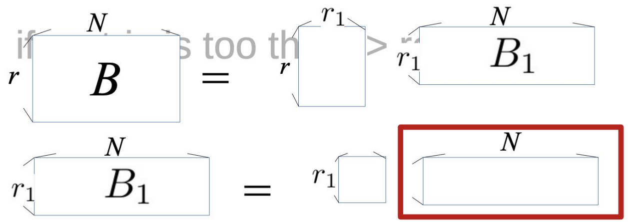
Low-rank SVD works but can we do better than that?

Two key Ideas:

recursively applying low-rank SVD

1)

$$r(M+N) \le MN$$



gets thinner and thinner

Two ideas to do better

Low-rank SVD works but can we do better than that?

Two key Ideas:

recursively applying low-rank SVD

1)
$$r(M+N) \leq MN$$

2) if matrix is too thin => reshape

$$\frac{M}{m} + mN \leq M + N$$

Two ideas to do better

Low-rank SVD works but can we do better than that?

Two key Ideas:

recursively applying low-rank SVD

1) $$r(M+N) \leq MN$$

2) if matrix is too thin => reshape

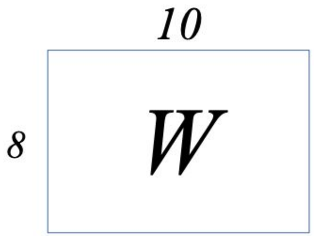
$$\frac{M}{m} + mN \leq M + N$$

Given two matrices X,Y with fixed total elements XY = C
Min when X~=Y; func -> C/X+X
Eg - 50+2 = 52
25+4 = 29 (less than 52)

Tensor-Train Decomposition

Combination of the two ideas we discussed

If we want to TT-decompose a matrix

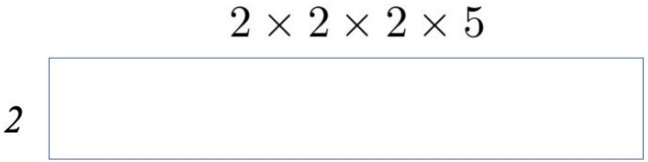


First, need to reshape it into a

tensor $\mathcal{W} : 2 \times 2 \times 2 \times 2 \times 5$

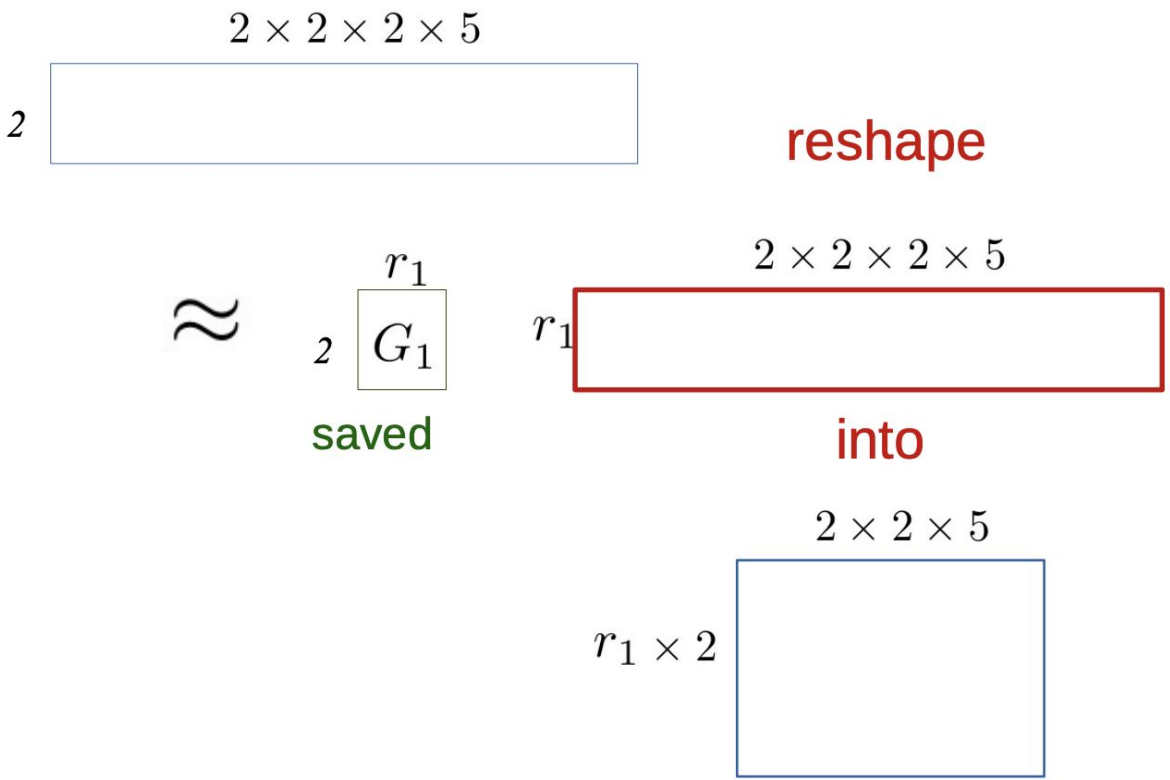
Tensor-Train Decomposition

unfold $\mathcal{W}$ by 1st dimension

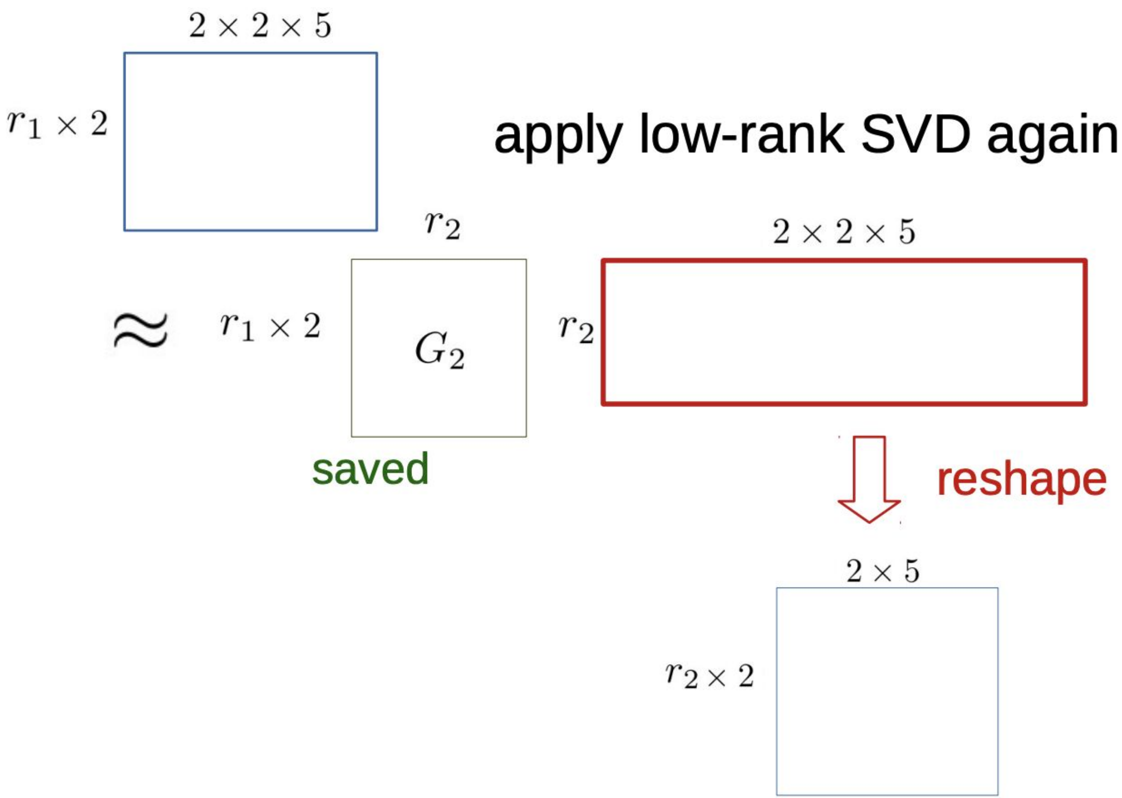


and apply low-rank SVD

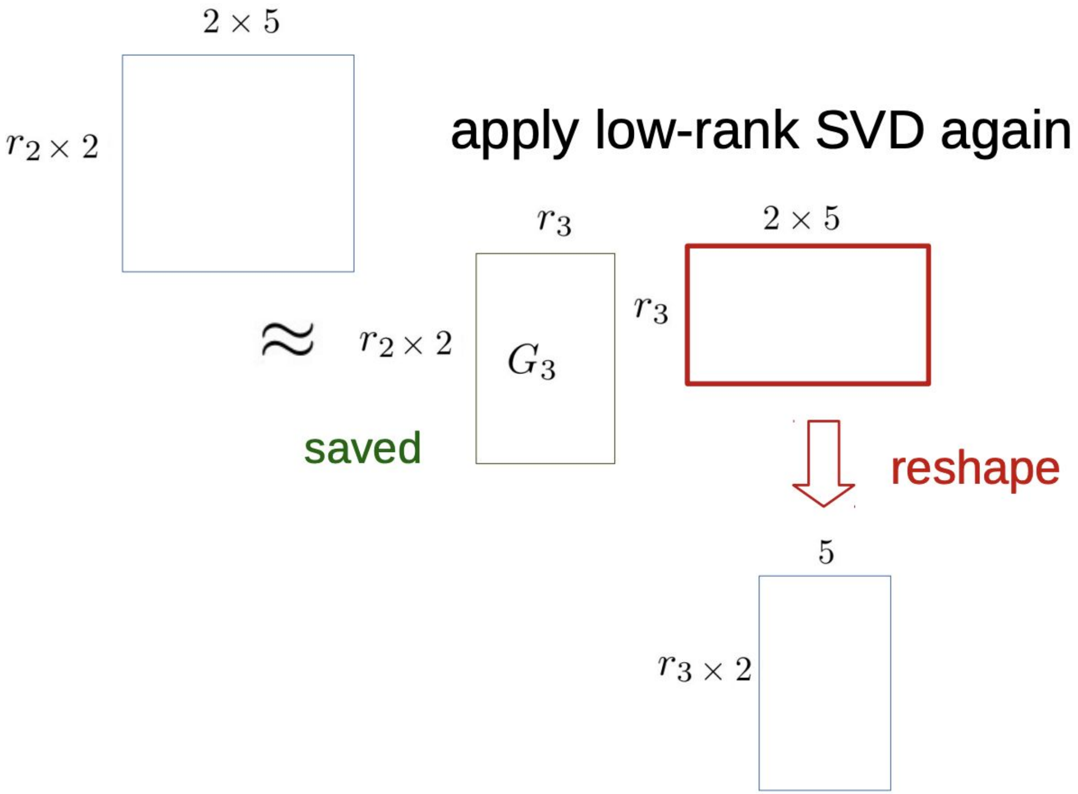
Tensor-Train Decomposition



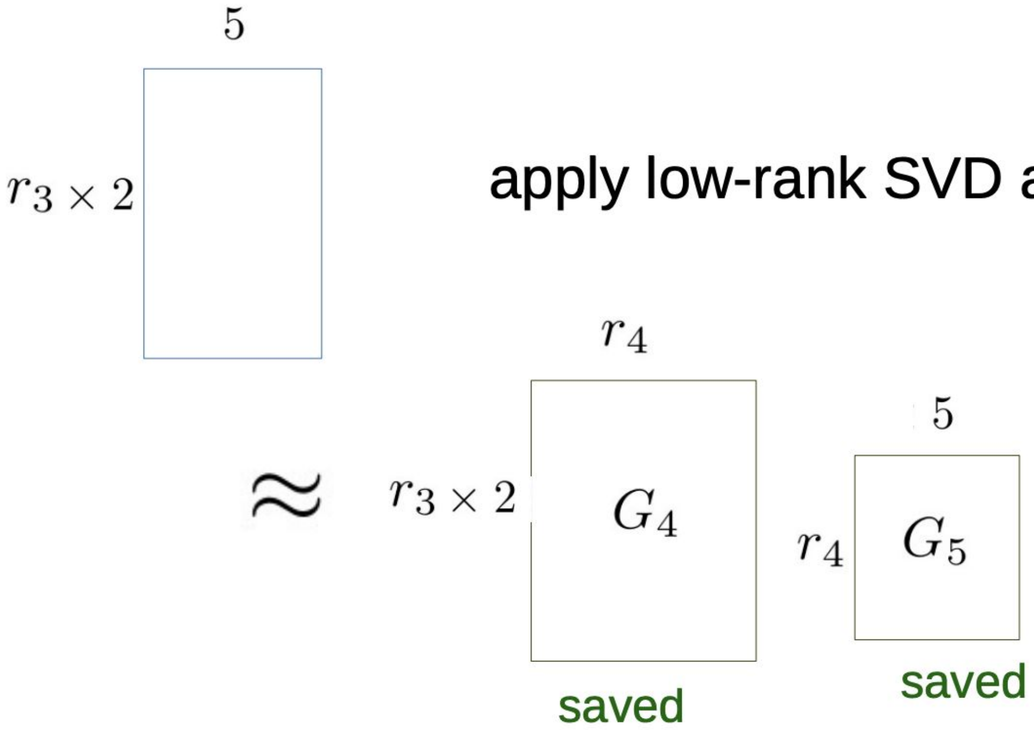
Tensor-Train Decomposition



Tensor-Train Decomposition

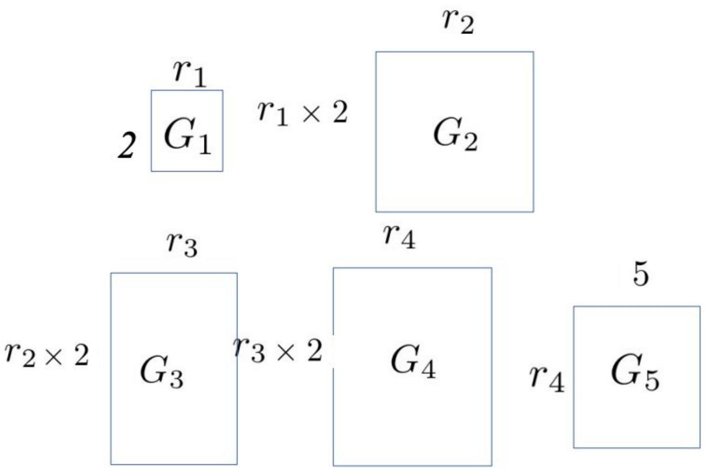


Tensor-Train Decomposition



apply low-rank SVD again

Tensor-Train Decomposition

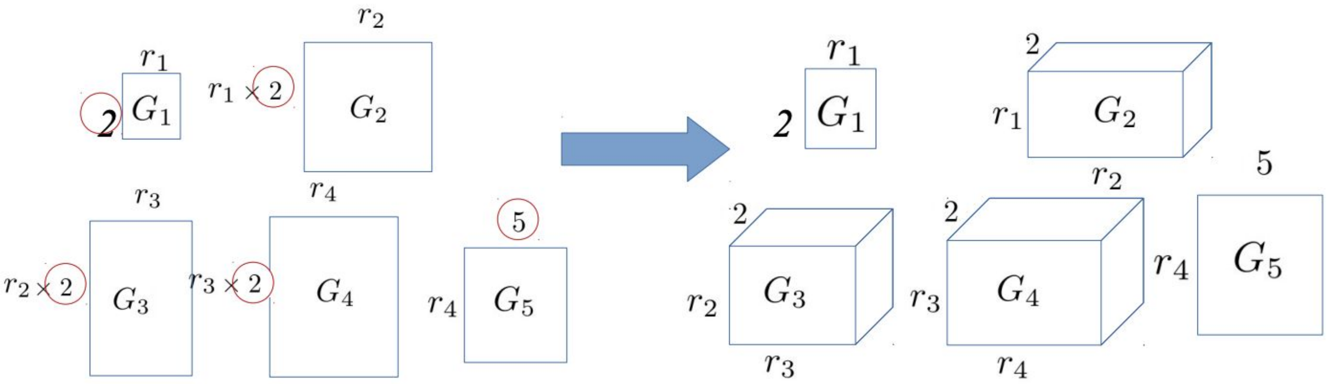


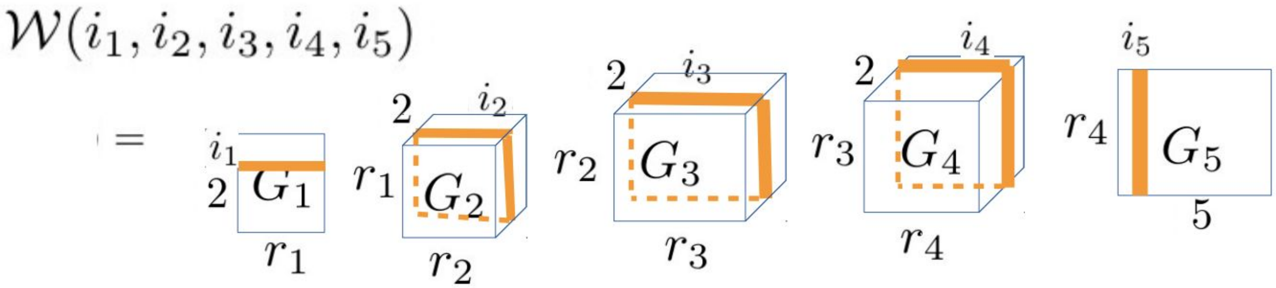
can approximate

$$\mathcal{W} : 2 \times 2 \times 2 \times 2 \times 5$$

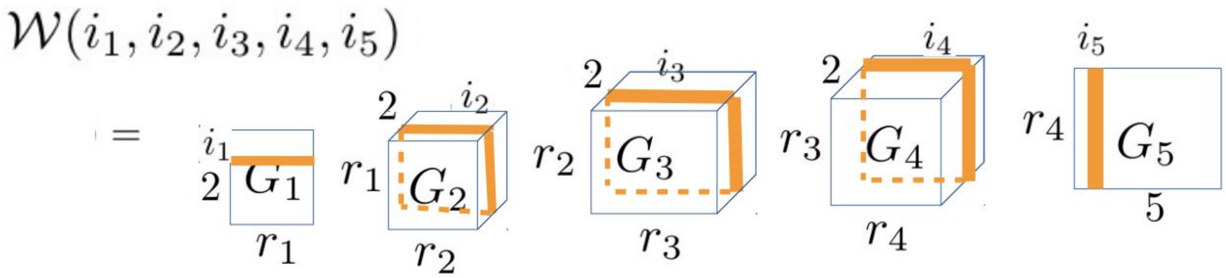
Tensor-Train Decomposition

fold into core tensors





Tensor-Train Decomposition



$$\mathcal{A}(\boldsymbol{i}) = G_1[i_1]G_2[i_2]\cdots G_d[i_d]$$

Tensor-Train format

TT-rank = SVD decomposition rank

$$(r_1, r_2, r_3, r_4, r_5)$$

TT-SVD algorithm for TT-Decomposition

Suppose, we want to approximate:

$$A(i_1, \dots, i_d) \approx G_1(i_1)G_2(i_2)G_3(i_3)G_4(i_4)$$

1. A_1 is an $n_1 \times (n_2 n_3 n_4)$ reshape of A.
2. $U_1, S_1, V_1 = \mathrm{SVD}(A_1)$, U_1 is $n_1 \times r_1$ — first core
3. $A_2 = S_1 V_1^*$, A_2 is $r_1 \times (n_2 n_3 n_4)$.
   Reshape it into a $(r_1 n_2) \times (n_3 n_4)$ matrix
4. Compute its SVD:
   $U_2, S_2, V_2 = \mathrm{SVD}(A_2)$,
   U_2 is $(r_1 n_2) \times r_2$ — second core, V_2 is $r_2 \times (n_3 n_4)$
5. $A_3 = S_2 V_2^*$,
6. Compute its SVD:
   $U_3 S_3 V_3 = \mathrm{SVD}(A_3)$, U_3 is $(r_2 n_3) \times r_3$, V_3 is $r_3 \times n_4$

Sk

Properties of TT-decomposition

- Has been shown that for an arbitrary tensor A a TT-representation exists but is not unique.
  - It’s natural to seek a representation with the lowest ranks
- TT-representation is very efficient in terms of memory if ranks are small

$$\sum_{k=1}^{d} n_k \, \mathrm{r}_{k-1} \, \mathrm{r}_k \quad \text{vs} \quad \prod_{k=1}^{d} n_k$$

- Efficient rounding to prevent explosion of TT-Ranks
- Efficiently perform several types of operations on tensors
  - Addition/ multiplication of constant
  - Summation and the entrywise product of tensors (results TT-tensors with more rank)
  - Global characteristics - sum of all elements and the Frobenius norm
  - Sum two TT-matrices
  - Matrix-by-vector (matrix-by-matrix) product

Properties of TT-decomposition

Operation	Output rank	Complexity
$\mathbf{A} \cdot \text{const}$	r_A	$O(dr_A))$
$\mathbf{A} + \text{const}$	$r_A + 1$	$O(dnr_A^2))$
$\mathbf{A} + \mathbf{B}$	$r_A + r_B$	$O(dn(r_A + r_B)^2)$
$\mathbf{A} \odot \mathbf{B}$	$r_A r_B$	$O(dnr_A^2 r_B^2)$
$\mathrm{sum}(\mathbf{A})$	$-$	$O(dnr_A^2)$
$\ldots$		

Tensor network diagrams

N-index tensor = shape with N lines

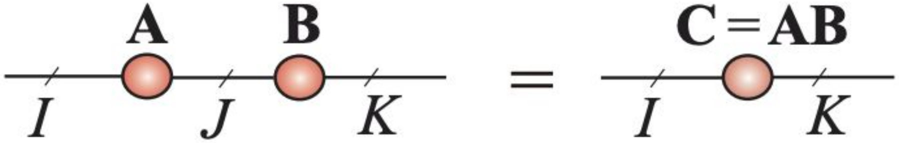
$$T^{s_1 s_2 s_3 \cdots s_N} =$$

Low-order tensor examples

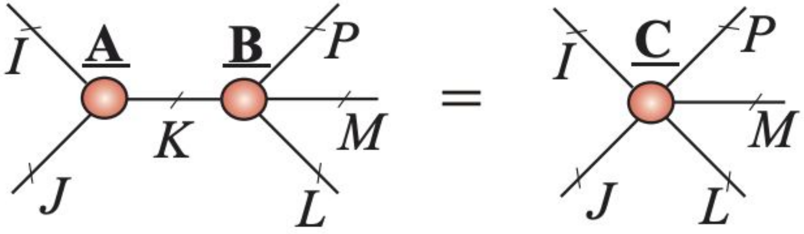
v_j M_{ij} T_{ijk}

- Matrix-vector multiplication

- Matrix-matrix multiplication



- Tensor contraction



$$\sum_{k=1}^{K} a_{i,j,k} \, b_{k,l,m,p} = c_{i,j,l,m,p}$$

Formal definition: TT Decomposition of Tensor

Tensor **A** can be decomposed to TT format as:

$$\mathbf{A}(i_1, i_2, \ldots, i_d) = \mathbf{G_1}[i_1]\mathbf{G_2}[i_2]\ldots\mathbf{G_d}[i_d]$$

Where:

$$\mathbf{G_k}[i_k] \in \mathbb{R}^{r_{k-1} \times r_k} \quad , \quad r_0 = r_d = 1$$

Compression:

$$O\left(n^d\right) \to O\left(ndr^2\right)$$

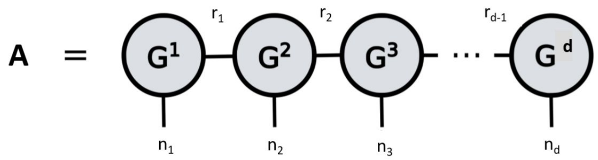
- TT-cores: $\mathbf{G_k}$
- TT-ranks: r_k
- TT max rank $r = max\ r_k \quad , \quad k = 0, \ldots, d$

$$\sum_{k=1}^{d} n_k r_{k-1} r_k = O(ndr^2)$$

Formal definition: TT Decomposition of Tensor

Tensor **A** can be decomposed to TT format as:

$$\mathbf{A}(i_1, i_2, \ldots, i_d) = \mathbf{G_1}[i_1]\mathbf{G_2}[i_2]\ldots\mathbf{G_d}[i_d]$$



Where:

$$\mathbf{G_k}[i_k] \in \mathbb{R}^{r_{k-1} \times r_k} \quad , \quad r_0 = r_d = 1$$

Compression:

$$O\left(n^d\right) \to O\left(ndr^2\right)$$

- TT-cores: $\mathbf{G_k}$
- TT-ranks: r_k
- TT max rank $r = max\ r_k \quad , \quad k = 0, \ldots, d$

$$\sum_{k=1}^{d} n_k r_{k-1} r_k = O(ndr^2)$$

Formal definition: TT-Vector

Consider a vector b: $b \in \mathbb{R}^{\mathrm{N}}$

Where: $N = \Pi_{k=1}^{d} n_k$

We can establish a bijection:

$$\mu : l \in \{1, \ldots, N\} \mapsto (\mu_1(l), \ldots, \mu_d(l))$$

Where:

$$\mu_k(l) \in \{1, \ldots, n_k\}$$

We can represent it using a tensor B:

$$B \in \mathbb{R}^{\mathrm{n_1 x n_2 x \ldots x n_d}}$$

Where:

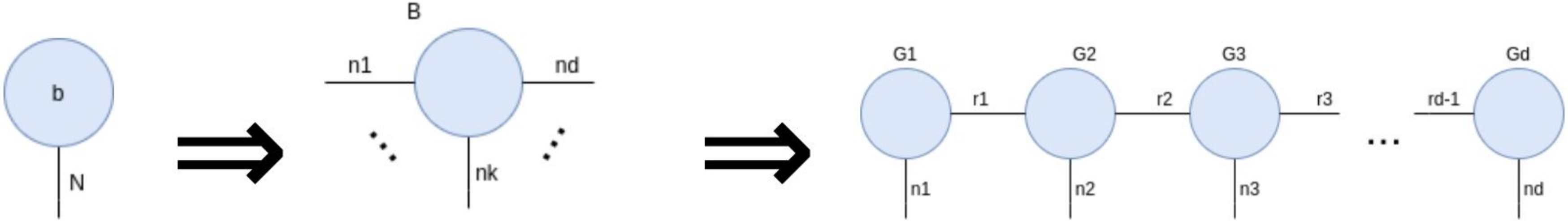
$$B((\mu_1(l), \ldots, \mu_d(l)) = b_l$$

Step1: Convert the large matrix into a tensor

Step2: Decompose into TT-representation to get a TT-vector

Formal definition: TT-Vector network diagram

$$b(l) = B((\mu_1(l), \ldots, \mu_d(l)) = \underbrace{G_1[\mu_1(l)]}_{1Xr_1}\underbrace{G_2[\mu_2(l)]}_{r_1Xr_2}\ldots\underbrace{G_d[\mu_d(l)]}_{r_{d-1}X1}$$



Formal definition: TT-Matrix

Consider a matrix A:

Where: $A \in \mathbb{R}^{\mathrm{MxN}}$

And: $M = \Pi_{k=1}^{d} m_k$, $N = \Pi_{k=1}^{d} n_k$

We can establish the bijections:

$$\nu : t \in \{1, \ldots, M\} \mapsto (\nu_1(t), \ldots, \nu_d(t))$$

And:

$$\mu : l \in \{1, \ldots, N\} \mapsto (\mu_1(l), \ldots, \mu_d(l))$$

Where:

$$\boldsymbol{\nu}(t) = (\nu_1(t), \ldots, \nu_d(t)) \text{ and } \boldsymbol{\mu}(\ell) = (\mu_1(\ell), \ldots, \mu_d(\ell))$$

Formal definition: TT-Matrix

We can represent using a tensor W:

$$W \in \mathbb{R}^{\mathrm{m_1 n_1 x m_2 n_2 x \ldots x m_d n_d}}$$

Where:

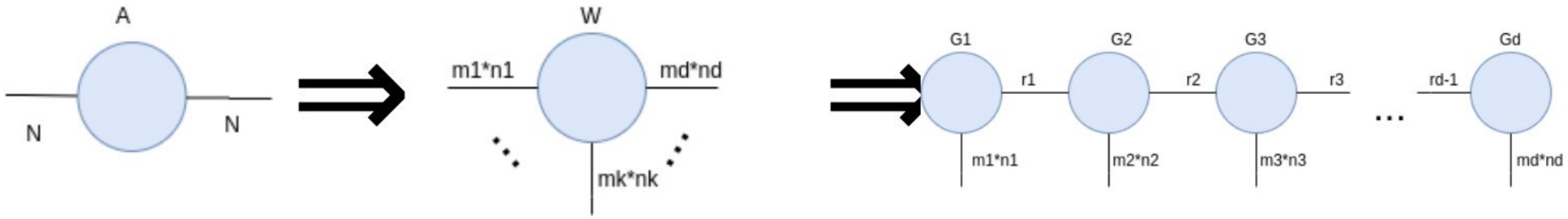
$$W(t, \ell) = \mathcal{W}((\nu_1(t), \mu_1(\ell)), \ldots, (\nu_d(t), \mu_d(\ell)))$$

Cores: $\boldsymbol{G}_k[\nu_k(t), \mu_k(\ell)],\ k = 1, \ldots, d,$

Index: $(\nu_k(t), \mu_k(\ell))$

Formal definition: TT-Matrix network diagram

$$A(t,l) = W((\nu_1(t),\mu_1(l)),\ldots,(\nu_d(t),\mu_d(l))) = \underbrace{G_1[\nu_1(t),\mu_1(l)]}_{1Xr_1}\underbrace{G_2[\nu_2(t),\mu_2(l)]}_{r_1Xr_2}\ldots\underbrace{G_d[\nu_d(t),\mu_d(l)]}_{r_{d-1}X1}$$



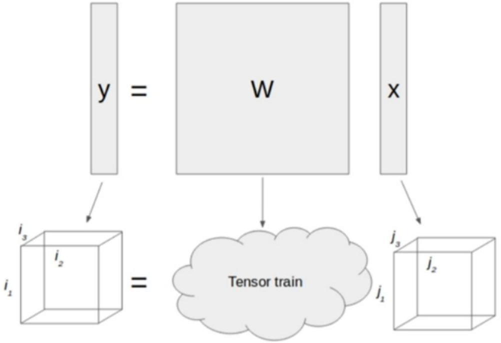
TensorNet:

TT-layer is a fully- connected layer with the weight matrix stored in the TT-format.

- A neural network with **one or more TT-layers as TensorNet**.

FC-layer:

$$\underbrace{\mathbf{y}}_{M} = \underbrace{\mathbf{W}}_{M \times N} \underbrace{\mathbf{x}}_{N} + \underbrace{\mathbf{b}}_{M}$$



TT-Layer

$$\mathcal{Y}(i_1, \ldots, i_d) = \sum_{j_1, \ldots, j_d} \boldsymbol{G}_1[i_1, j_1] \ldots \boldsymbol{G}_d[i_d, j_d] \, \mathcal{X}(j_1, \ldots, j_d) + \mathcal{B}(i_1, \ldots, i_d).$$

A TT-layer transforms a d-dimensional tensor $\boldsymbol{\mathcal{X}}$ (formed from the corresponding vector $\boldsymbol{x}$) to the d-dimensional tensor $\boldsymbol{\mathcal{Y}}$ (which correspond to the output vector $\boldsymbol{y}$). We assume that the weight matrix $\boldsymbol{W}$ is represented in the TT-format with the cores $\boldsymbol{G}_k[i_k, j_k]$.

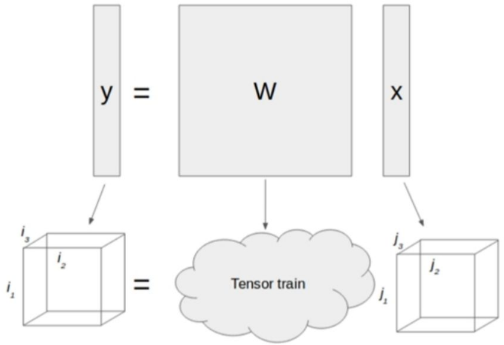
TensorNet:

TT-layer is a fully- connected layer with the weight matrix stored in the TT-format.

- A neural network with **one or more TT-layers as TensorNet**.

FC-layer:

$$\underbrace{\mathbf{y}}_{M} = \underbrace{\mathbf{W}}_{M \times N} \underbrace{\mathbf{x}}_{N} + \underbrace{\mathbf{b}}_{M}$$



TT-Layer

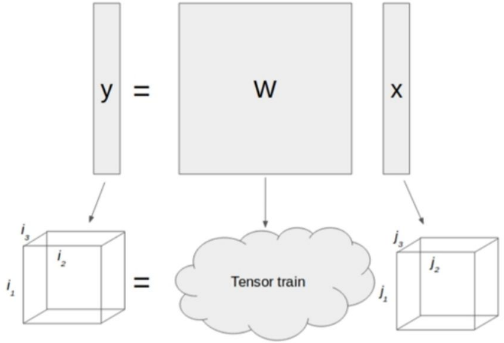
$$\mathcal{Y}(i_1, \ldots, i_d) = \sum_{j_1, \ldots, j_d} \boldsymbol{G}_1[i_1, j_1] \ldots \boldsymbol{G}_d[i_d, j_d] \, \mathcal{X}(j_1, \ldots, j_d) + \mathcal{B}(i_1, \ldots, i_d).$$

TensorNet:

TT-layer is a fully- connected layer with the weight matrix stored in the TT-format.

- A neural network with **one or more TT-layers as TensorNet**.

FC-layer:



$$\underbrace{\mathbf{y}}_{M} = \underbrace{\mathbf{W}}_{M \times N} \underbrace{\mathbf{x}}_{N} + \underbrace{\mathbf{b}}_{M}$$

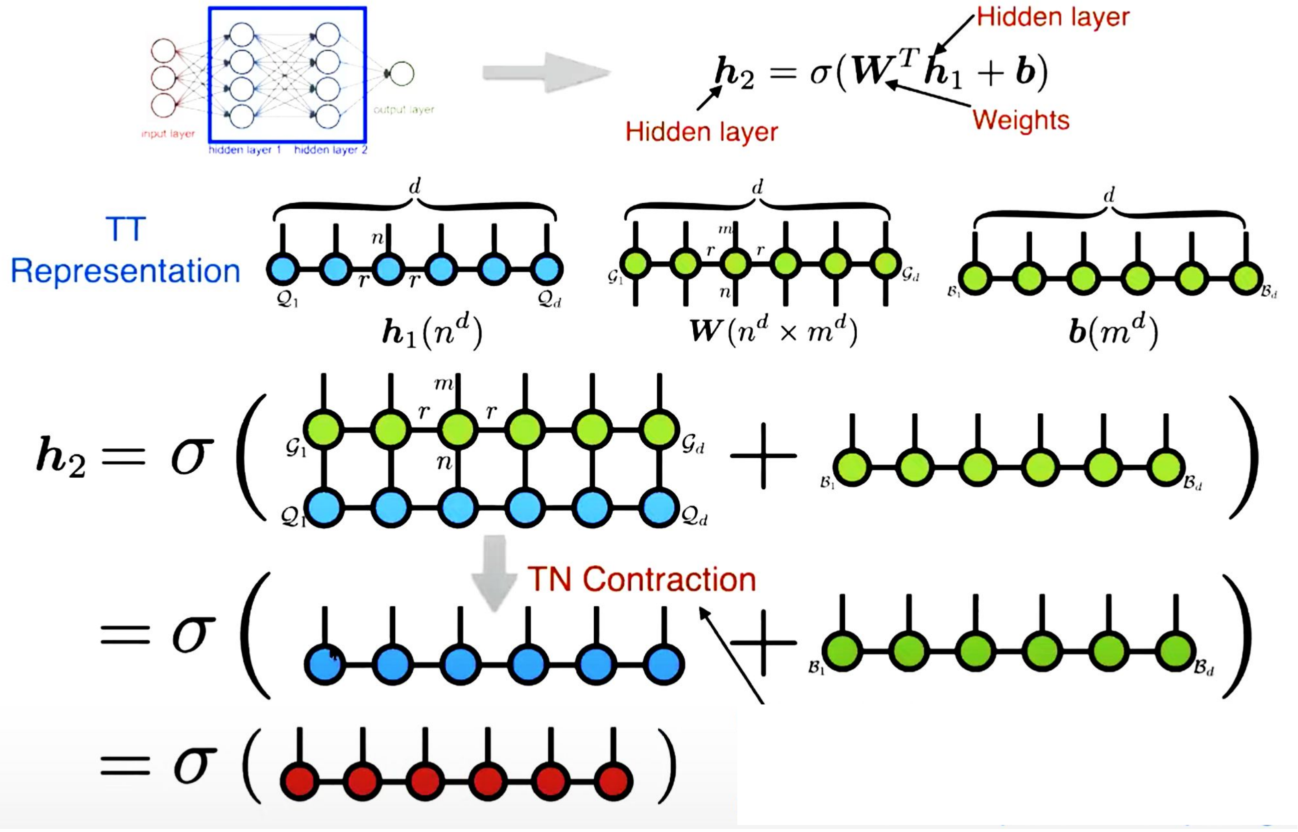
TT-Layer

$$\mathcal{Y}(i_1, \dots, i_d) = \sum_{j_1, \dots, j_d} \boldsymbol{G}_1[i_1, j_1] \dots \boldsymbol{G}_d[i_d, j_d] \, \mathcal{X}(j_1, \dots, j_d) + \mathcal{B}(i_1, \dots, i_d).$$

No of cores
Maximal rank
max (m_k)

Forward pass: $O(M\,N) \rightarrow O(d r^2 \, m \, \max\{M, N\})$

TensorNet network diagram



Backpropagation

$$\frac{\partial L}{\partial \boldsymbol{x}} = \boldsymbol{W}^{\intercal} \frac{\partial L}{\partial \boldsymbol{y}}, \qquad \boxed{\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{y}} \boldsymbol{x}^{\intercal},} \qquad \frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{y}}.$$

layer. To compute the gradient of the loss function w.r.t. the bias vector $\boldsymbol{b}$ and w.r.t. the input vector $\boldsymbol{x}$ one can use equations (6). The latter can be applied using the matrix-by-vector product (where the matrix is in the TT-format) with the complexity of $O(dr^2 n \max\{m, n\}^d) = O(dr^2 n \max\{M, N\})$.

$$\underbrace{\frac{\partial L}{\partial \mathbf{G}_k[\tilde{i}_k, \tilde{j}_k]}}_{\mathrm{r}_{k-1} \times \mathrm{r}_k} = \sum_{\boldsymbol{i}} \frac{\partial L}{\partial \mathcal{Y}(\boldsymbol{i})} \frac{\partial \mathcal{Y}(\boldsymbol{i})}{\partial \mathbf{G}_k[\tilde{i}_k, \tilde{j}_k]}. \qquad O(M\, \mathrm{r}_{k-1}\, \mathrm{r}_k)$$

Backpropagation

$$\mathcal{Y}(\boldsymbol{i}) = \sum_{\boldsymbol{j}} G_1\underbrace{[i_1, j_1]\cdots}_{a^T} \underset{X}{G_k[i_k, j_k]} \underbrace{\cdots G_d[i_d, j_d]}_{b} \mathcal{X}(\boldsymbol{j}) + \mathcal{B}(\boldsymbol{i})$$

$$\frac{\partial \mathcal{Y}(\boldsymbol{i})}{\partial G_k[i_k, j_k]}$$

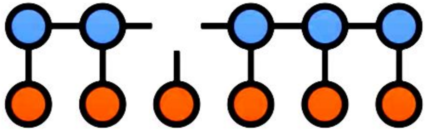
calculus rule

$$\boxed{\frac{\partial (a^T X b)}{\partial X} = a b^T}$$

Backpropagation

$$\mathcal{Y}(\boldsymbol{i}) = \sum_{\boldsymbol{j}} G_1[i_1, j_1] \cdots G_k[i_k, j_k] \cdots G_d[i_d, j_d] \mathcal{X}(\boldsymbol{j}) + \mathcal{B}(\boldsymbol{i})$$

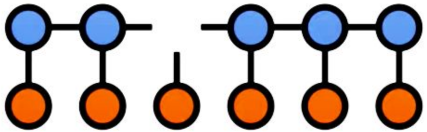
$$\frac{\partial \mathcal{Y}(\boldsymbol{i})}{\partial G_k[i_k, j_k]} = \sum_{\boldsymbol{j} \backslash j_k} \left(G_1[i_1, j_1] \cdots\right)^T \left(\cdots G_d[i_d, j_d]\right)^T \mathcal{X}(\boldsymbol{j})$$



Backpropagation

$$\mathcal{Y}(\boldsymbol{i}) = \sum_{\boldsymbol{j}} G_1[i_1, j_1] \cdots G_k[i_k, j_k] \cdots G_d[i_d, j_d] \mathcal{X}(\boldsymbol{j}) + \mathcal{B}(\boldsymbol{i})$$

$$\frac{\partial \mathcal{Y}(\boldsymbol{i})}{\partial G_k[i_k, j_k]} = \sum_{\boldsymbol{j} \setminus j_k} \left(G_1[i_1, j_1] \cdots\right)^T \left(\cdots G_d[i_d, j_d]\right)^T \mathcal{X}(\boldsymbol{j})$$



$$O(dr^2 m \max\{M, N\})$$

backward pass

$$O(MN) \to O(d^2 r^4 m \max\{M, N\})$$

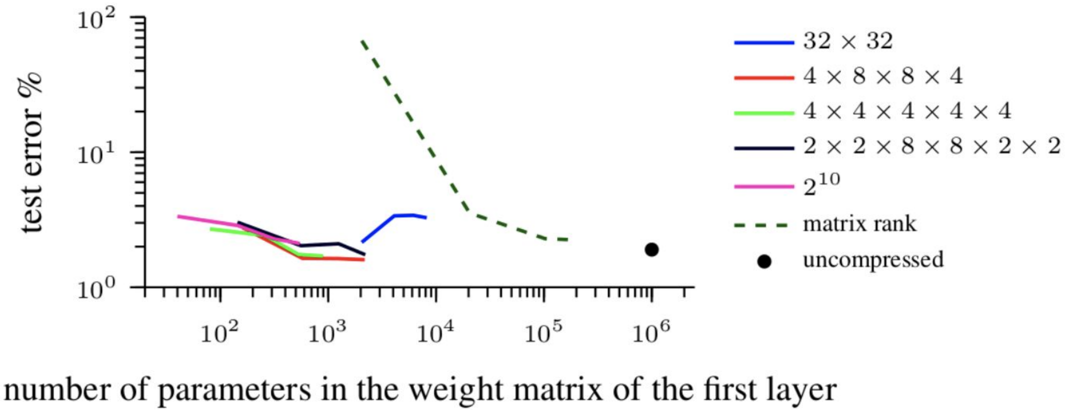
Experimental results: MNIST

- Small: MNIST

FC 1024 x 10

FC 1024 x 1024

compress 1st layer



TT-Layers provide much better flexibility than the matrix rank keeping same compression level

TT-layers with too small number of values for each tensor dimension and with too few dimensions **perform worse than their more balanced counterparts**

Experimental results: ImageNet

Architecture	TT-layers compr.	vgg-16 compr.	vgg-19 compr.	vgg-16 top 1	vgg-16 top 5	vgg-19 top 1	vgg-19 top 5
FC FC FC	1	1	1	30.9	11.2	29.0	10.1
TT4 FC FC	50 972	3.9	3.5	31.2	11.2	29.8	10.4
TT2 FC FC	194 622	3.9	3.5	31.5	11.5	30.4	10.9
TT1 FC FC	713 614	3.9	3.5	33.3	12.8	31.9	11.8
TT4 TT4 FC	37 732	7.4	6	32.2	12.3	31.6	11.7
MR1 FC FC	3 521	3.9	3.5	99.5	97.6	99.8	99
MR5 FC FC	704	3.9	3.5	81.7	53.9	79.1	52.4
MR50 FC FC	70	3.7	3.4	36.7	14.9	34.5	15.8

Table 2: Substituting the fully-connected layers with the TT-layers in vgg-16 and vgg-19 networks on the ImageNet dataset. FC stands for a fully-connected layer; TT□ stands for a TT-layer with all the TT-ranks equal "□"; MR□ stands for a fully-connected layer with the matrix rank restricted to "□". We report the compression rate of the TT-layers matrices and of the whole network in the second, third and fourth columns.

Great compression factor of 194622 with 0.3 accuracy drop

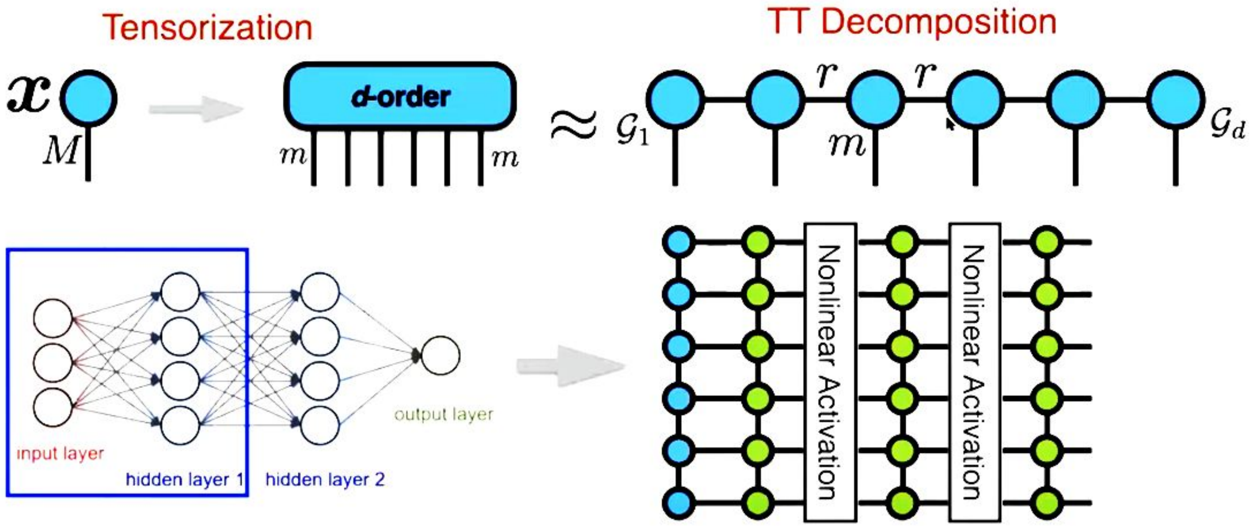
Experimental results: ImageNet

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

Table 3: Inference time for a 25088×4096 fully-connected layer and its corresponding TT-layer with all the TT-ranks equal 4. The memory usage for feeding forward one image is 392MB for the fully-connected layer and 0.766MB for the TT-layer.

TT-Layer has better inference time in comparison to FC-Layer

Challenges



- Input data may not admit low-rank TT approximation (small r)
- Nonlinear activation destroy TT format

Similar works

Lebedev V. et al. Speeding-up convolutional neural networks using fine-tuned cp-decomposition arXiv:1412.6553.

8.5x speedup with 1% accuracy drop

Recent example: Yang, Yinchong, Denis Krompass, and Volker Tresp. "Tensor-Train Recurrent Neural Networks for Video Classification." arXiv:1707.01786

3000 parameters in TT-LSTM vs 71,884,800 in LSTM

Accuracy is better due to additional regularisation

Thanks

References

[1] Tensorizing Neural Network; NIPS 2015 slides [link]

[2] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, Dmitry Vetrov, Tensorizing Neural Networks; NIPS 2015

[3] Slides by Moussa Traore Mehraveh Javan [link]

[4] Tensor Train in machine learning Slides [link]

[5] More slides -> [Link1], [Link2]

[6] Lecture Notes [Link]