Introduction to the Tensor Train Decomposition and Its Applications in Machine Learning

Anton Rodomanov

Higher School of Economics, Russia

Bayesian methods research group (http://bayesgroup.ru)

14 March 2016

HSE Seminar on Applied Linear Algebra, Moscow, Russia

Outline

1. Tensor Train Format
2. ML Application 1: Markov Random Fields
3. ML Application 2: TensorNet

What is a tensor?

Tensor = multidimensional array:

$$\mathbf{A} = [A(i_1, \dots, i_d)], \qquad i_k \in \{1, \dots, n_k\}.$$

Terminology:

- *dimensionality* $= d$ (number of indices).
- *size* $= n_1 \times \cdots \times n_d$ (number of nodes along each axis).

Case $d = 1 \Rightarrow$ vector, $d = 2 \Rightarrow$ matrix.

Curse of dimensionality

Number of elements $= n^d$ (exponential in d)
When $n = 2,\ d = 100$

$$2^{100} > 10^{30} \qquad (\approx 10^{18} \text{ PB of memory}).$$

Cannot work with tensors using standard methods.

Tensor rank decomposition [Hitchcock, 1927]

Recall the rank decomposition for matrices:

$$A(i_1, i_2) = \sum_{\alpha=1}^{r} U(i_1, \alpha) V(i_2, \alpha).$$

This can be generalized to tensors.

Tensor rank decomposition (canonical decomposition):

$$A(i_1, \ldots, i_d) = \sum_{\alpha=1}^{R} U_1(i_1, \alpha) \ldots U_d(i_d, \alpha).$$

The minimal possible R is called the (canonical) rank of the tensor **A**.

- (+) No curse of dimensionality.
- (-) Ill-posed problem [de Silva, Lim, 2008].
- (-) Rank R should be known in advance for many methods.
- (-) Computation of R is NP-hard [Hillar, Lim, 2013].

Unfolding matrices: definition

Every tensor $\mathbf{A}$ has $d-1$ unfolding matrices:

$$A_k := [A(i_1 \dots i_k; i_{k+1} \dots i_d)],$$

where

$$A(i_1 \dots i_k; i_{k+1} \dots i_d) := A(i_1, \dots, i_d).$$

Here $i_1 \dots i_k$ and $i_{k+1} \dots i_d$ are row and column (multi)indices; A_k are matrices of size $M_k \times N_k$ with $M_k = \prod_{s=1}^{k} n_s$, $N_k = \prod_{s=k+1}^{d} n_s$.

This is just a `reshape`.

Unfolding matrices: example

Consider $\mathbf{A} = [A(i,j,k)]$ given by its elements:

$$\begin{aligned} A(1,1,1) &= 111, & A(2,1,1) &= 211,\\ A(1,2,1) &= 121, & A(2,2,1) &= 221,\\ A(1,1,2) &= 112, & A(2,1,2) &= 212,\\ A(1,2,2) &= 122, & A(2,2,2) &= 222. \end{aligned}$$

Then

$$A_1 = [A(i;jk)] = \begin{bmatrix} 111 & 121 & 112 & 122 \\ 211 & 221 & 212 & 222 \end{bmatrix},$$

$$A_2 = [A(ij;k)] = \begin{bmatrix} 111 & 112 \\ 211 & 212 \\ 121 & 122 \\ 221 & 222 \end{bmatrix}.$$

Tensor Train decompositon: motivation

Main idea: variable splitting.

Consider a rank decomposition of an unfolding matrix:

$$A(i_1 i_2; i_3 i_4 i_5 i_6) = \sum_{\alpha_2} U(i_1 i_2; \alpha_2) V(i_3 i_4 i_5 i_6; \alpha_2).$$

On the left: 6-dimensional tensor; on the right: 3- and 5-dimensional.
The dimension has reduced!
Proceed recursively.

Tensor Train decomposition [Oseledets, 2011]

- TT-format for a tensor $\mathbf{A}$:
$$A(i_1, \dots, i_d) = \sum_{\alpha_0, \dots, \alpha_d} G_1(\alpha_0, i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \dots G_d(\alpha_{d-1}, i_d, \alpha_d).$$
- This can be written compactly as a matrix product:
$$A(i_1, \dots, i_d) = \underbrace{G_1[i_1]}_{1 \times r_1} \underbrace{G_2[i_2]}_{r_1 \times r_2} \dots \underbrace{G_d[i_d]}_{r_{d-1} \times 1}$$
- Terminology:
  - G_i: TT-cores (collections of matrices)
  - r_i: TT-ranks
  - $r = \max r_i$: maximal TT-rank
- TT-format uses $O(dnr^2)$ memory to store $O(n^d)$ elements.
- Efficient only if the ranks are small.

TT-format: example

- Consider a tensor:

$$A(i_1, i_2, i_3) := i_1 + i_2 + i_3,$$
$$i_1 \in \{1,2,3\}, \quad i_2 \in \{1,2,3,4\}, \quad i_3 \in \{1,2,3,4,5\}.$$

- Its TT-format:

$$A(i_1, i_2, i_3) = G_1[i_1] G_2[i_2] G_3[i_3],$$

where

$$G_1[i_1] := \begin{bmatrix} i_1 & 1 \end{bmatrix}, \quad G_2[i_2] := \begin{bmatrix} 1 & 0 \\ i_2 & 1 \end{bmatrix}, \quad G_3[i_3] := \begin{bmatrix} 1 \\ i_3 \end{bmatrix}$$

- Check:

$$A(i_1, i_2, i_3) = \begin{bmatrix} i_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ i_2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ i_3 \end{bmatrix} =$$
$$= \begin{bmatrix} i_1 + i_2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ i_3 \end{bmatrix} = i_1 + i_2 + i_3.$$

TT-format: example

- Consider a tensor:

$$A(i_1, i_2, i_3) := i_1 + i_2 + i_3,$$
$$i_1 \in \{1, 2, 3\}, \quad i_2 \in \{1, 2, 3, 4\}, \quad i_3 \in \{1, 2, 3, 4, 5\}.$$

- Its TT-format:

$$A(i_1, i_2, i_3) = G_1[i_1] G_2[i_2] G_3[i_3],$$

where

$$G_1 = \left(\begin{bmatrix} 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 1 \end{bmatrix}, \begin{bmatrix} 3 & 1 \end{bmatrix} \right)$$
$$G_2 = \left(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} \right)$$
$$G_3 = \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \begin{bmatrix} 1 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 \\ 5 \end{bmatrix} \right)$$

- The tensor has $3 \cdot 4 \cdot 5 = 60$ elements.
  TT-format uses 32 elements to describe it.

Finding a TT-representation of a tensor

General ways of building a TT-decomposition of a tensor:

- Analytical formulas for the TT-cores.
- TT-SVD algorithm [Oseledets, 2011]:
  - Exact quasi-optimal method.
  - Suitable only for small tensors (which fit into memory).
- Interpolation algorithms: AMEn-cross [Dolgov & Savostyanov, 2013], DMRG [Khoromskij & Oseledets, 2010], TT-cross [Oseledets, 2010]
  - Approximate heuristically-based methods.
  - Can be applied for large tensors.
  - No strong guarantees but work well in practice.
- Operations between other tensors in the TT-format: addition, element-wise product etc.

TT-interpolation: problem formulation

Input: procedure $A(i_1, \dots, i_d)$ for computing an arbitrary element of $\mathbf{A}$.

Output: TT-decomposition of $\mathbf{B} \approx \mathbf{A}$:

$$B(i_1, \dots, i_d) = G_1[i_1] \dots G_d[i_d].$$

Matrix interpolation: Matrix-cross

Let $\mathbf{A} \in \mathbf{R}^{m \times n}$ with rank r.
It admits a skeleton decomposition [Goreinov et al., 1997]:

$$\mathbf{A} = \underbrace{\mathbf{C}}_{m \times r} \underbrace{\hat{\mathbf{A}}^{-1}}_{r \times r} \underbrace{\mathbf{R}}_{r \times n},$$

where

- $\hat{\mathbf{A}} = \mathbf{A}(\mathcal{I}, \mathcal{J})$: non-singular matrix.
- $\mathbf{C} = \mathbf{A}(:, \mathcal{J})$: columns containing $\hat{\mathbf{A}}$.
- $\mathbf{R} = \mathbf{A}(\mathcal{I}, :)$: rows containing $\hat{\mathbf{A}}$.

Q: Which $\hat{\mathbf{A}}$ to choose?
A: Any non-singular submatrix if $\operatorname{rank} \mathbf{A} = r \Rightarrow$ exact decomposition.

Q: What if $\operatorname{rank} \mathbf{A} \approx r$? Different $\hat{\mathbf{A}}$ will give different error.
A: Choose a maximal volume submatrix [Goreinov, Tyrtyshnikov, 2001]. It can be found with the maxvol algorithm [Goreinov et al., 2008].

TT interpolation with TT-cross: example

Hilbert tensor:

$$A(i_1, i_2, \ldots, i_d) := \frac{1}{i_1 + i_2 + \ldots + i_d}.$$

TT-rank	Time	Iterations	Relative accuracy
2	1.37	5	1.897278e+00
3	4.22	7	5.949094e-02
4	7.19	7	2.226874e-02
5	15.42	9	2.706828e-03
6	21.82	9	1.782433e-04
7	29.62	9	2.151107e-05
8	38.12	9	4.650634e-06
9	48.97	9	5.233465e-07
10	59.14	9	6.552869e-08
11	72.14	9	7.915633e-09
12	75.27	8	2.814507e-09

[Oseledets & Tyrtyshnikov, 2009]

Operations: addition and multiplication by number

- Let $\mathbf{C} = \mathbf{A} + \mathbf{B}$:
  $$C(i_1, \dots, i_d) = A(i_1, \dots, i_d) + B(i_1, \dots, i_d).$$
  TT-cores of $\mathbf{C}$ are as follows:
  $$C_k[i_k] = \begin{bmatrix} A_k[i_k] & 0 \\ 0 & B_k[i_k] \end{bmatrix}, \quad k = 2, \dots, d-1,$$
  $$C_1[i_1] = \begin{bmatrix} A_1[i_1] & B_1[i_1] \end{bmatrix}, \quad C_d[i_d] = \begin{bmatrix} A_d[i_d] \\ B_d[i_d] \end{bmatrix}.$$
  The ranks are doubled.
- Multiplication by a number: $\mathbf{C} = \mathbf{A} \cdot \mathrm{const}$.
  Multiply only one core by const. The ranks do not increase.

Operations: element-wise product

Let $\mathbf{C} = \mathbf{A} \odot \mathbf{B}$:

$$C(i_1, \dots, i_d) = A(i_1, \dots, i_d) \cdot B(i_1, \dots, i_d).$$

TT-cores of $\mathbf{C}$ can be computed as follows:

$$C_k[i_k] = A_k[i_k] \otimes B_k[i_k],$$

where $\otimes$ is the Kronecker product operation.

$\operatorname{rank}(\mathbf{C}) = \operatorname{rank}(\mathbf{A}) \operatorname{rank}(\mathbf{B})$.

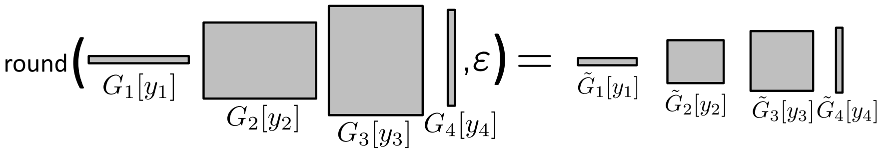
TT-format: efficient operations

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

TT-rounding

TT-rounding procedure: $\tilde{\mathbf{A}} = \text{round}(\mathbf{A}, \varepsilon)$, $\varepsilon =$ accuracy:

1. Maximally reduces TT-ranks ensuring that $\|\mathbf{A} - \tilde{\mathbf{A}}\|_F \leq \varepsilon \|\mathbf{A}\|_F$.
2. Uses SVD for compression (like TT-SVD).



Allows one to trade off accuracy vs maximal rank of the TT-representation.

Example: $\text{round}(\mathbf{A} + \mathbf{A}, \varepsilon_{\text{mach}}) = \mathbf{A}$ within machine accuracy $\varepsilon_{\text{mach}}$.

Example: multivariate integration

$$I(d) := \int\limits_{[0,1]^d} \sin(x_1 + x_2 + \ldots + x_d)\, dx_1 dx_2 \ldots dx_d = \mathrm{Im}\left(\left(\frac{e^i - 1}{i}\right)^d\right).$$

Use Chebyshev quadrature with $n = 11$ nodes + TT-cross with $r = 2$.

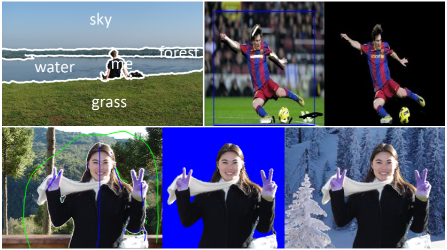
d	$I(d)$	Relative accuracy	Time
10	-6.299353e-01	1.409952e-15	0.14
100	-3.926795e-03	2.915654e-13	0.77
500	-7.287664e-10	2.370536e-12	4.64
1 000	-2.637513e-19	3.482065e-11	11.70
2 000	2.628834e-37	8.905594e-12	33.05
4 000	9.400335e-74	2.284085e-10	105.49

[Oseledets & Tyrtyshnikov, 2009]

Outline

1. Tensor Train Format
2. ML Application 1: Markov Random Fields
3. ML Application 2: TensorNet

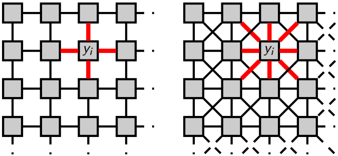
Motivational example: image segmentation



- **Task**: assign a label y_i to each pixel of an $M \times N$ image.
- Let $P(\mathbf{y})$ be the joint probability of labelling $\mathbf{y}$.
- Two extreme cases:
  - **No assumptions about independence**:
    - $O(K^{MN})$ parameters (K = total number of labels)
    - represents every distribution
    - intractable in general
  - **Everything is independent**: $P(\mathbf{y}) = p_1(y_1) \ldots p_{MN}(y_{MN})$
    - $O(MNK)$ parameters
    - represents only a small class of distributions
    - tractable

Graphical models

- Provide a convenient way to define probabilistic models using graphs.
- Two types: directed graphical models and Markov random fields.
- We will consider only (discrete) Markov random fields.
- The edges represent dependencies between the variables.
- E.g., for image segmentation:



A variable y_i is independent of the rest given its immediate neighbours.

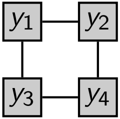
Markov random fields

- The model:

$$P(\mathbf{y}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \Psi_c(\mathbf{y}_c),$$

  - Z: normalization constant
  - $\mathcal{C}$: set of all (maximal) cliques in the graph
  - Ψ_c: non-negative functions which are called factors

- Example:



$$P(y_1, y_2, y_3, y_4) = \frac{1}{Z} \Psi_1(y_1)\Psi_2(y_2)\Psi_3(y_3)\Psi_4(y_4)$$
$$\times\, \Psi_{12}(y_1, y_2)\Psi_{24}(y_2, y_4)\Psi_{34}(y_3, y_4)\Psi_{13}(y_1, y_3)$$

The factors Ψ_{ij} measure 'compatibility' between variables y_i and y_j.

Main problems of interest

Probabilistic model:

$$P(\mathbf{y}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \Psi_c(\mathbf{y}_c) = \frac{1}{Z} \exp(-E(\mathbf{y})),$$

where E is the energy function:

$$E(\mathbf{y}) = \sum_{c \in \mathcal{C}} \Theta_c(\mathbf{y}_c), \qquad \Theta_c(\mathbf{y}_c) = -\ln \Psi_c(\mathbf{y}_c)$$

- Maximum a posteriori (MAP) inference:

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y}} P(\mathbf{y}) = \operatorname*{argmin}_{\mathbf{y}} E(\mathbf{y})$$

- Estimation of the normalization constant:

$$Z = \sum_{\mathbf{y}} P(\mathbf{y})$$

- Estimation of the marginal distributions:

$$P(y_i) = \sum_{\mathbf{y} \setminus y_i} P(\mathbf{y})$$

Tensorial perspective

- Energy and unnormalized probability are tensors:

$$\left.\begin{aligned} \mathbf{E}(y_1, \dots, y_n) &= \sum_{c=1}^{m} \boldsymbol{\Theta}_c(\mathbf{y}_c), \\ \widehat{\mathbf{P}}(y_1, \dots, y_n) &= \prod_{c=1}^{m} \boldsymbol{\Psi}_c(\mathbf{y}_c), \end{aligned}\right\} \quad \text{tensors}$$

  where $y_i \in \{1, \dots, d\}$.
- In this language:
  - MAP-inference $\iff$ minimal element in $\boldsymbol{E}$
  - Normalization constant $\iff$ sum of all the elements of $\widehat{\boldsymbol{P}}$

Details: **Putting MRFs on a Tensor Train [Novikov et al., ICML 2014]**.

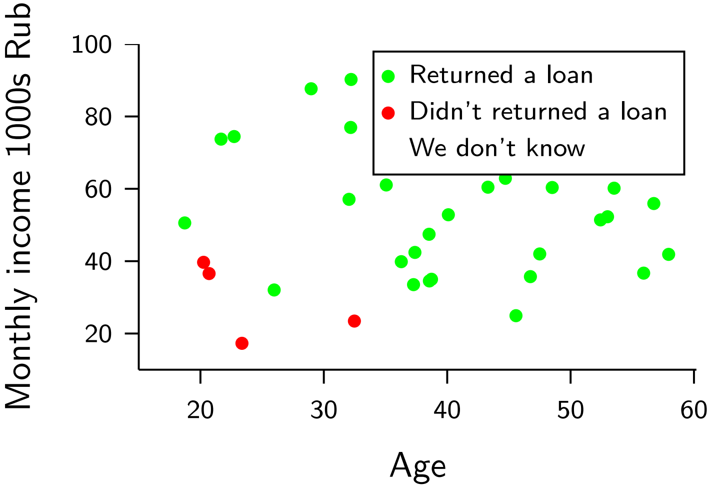
Outline

1 Tensor Train Format

2 ML Application 1: Markov Random Fields

3 ML Application 2: TensorNet

Classification problem



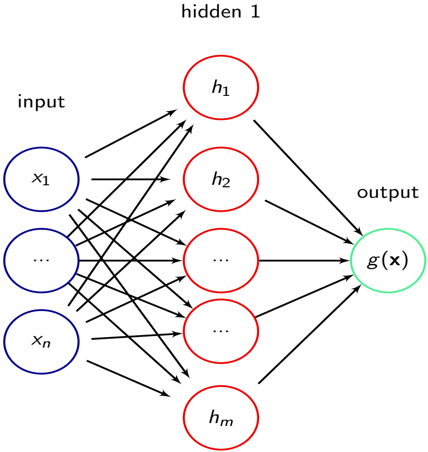
Neural networks

Use a composite function:

$$g(\mathbf{x}) = f(\mathbf{W}_2 \cdot \underbrace{f(\mathbf{W}_1 \mathbf{x})}_{\mathbf{h} \in \mathbf{R}^m})$$

$$h_k = f((\mathbf{W}_1 \mathbf{x})_k)$$

$$f(x) = \frac{1}{1 + \exp(-x)}$$



TensorNet: motivation

Goal: compress fully-connected layers.

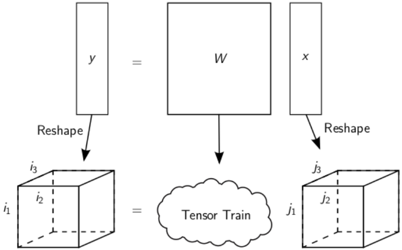
Why care about memory?

- State-of-the-art deep neural networks don't fit into the memory of mobile devices.
- Up to 95% of the parameters are in the fully-connected layers.
- A shallow network with a huge fully-connected layer can achieve almost the same accuracy as an ensemble of deep CNNs [Ba and Caruana, 2014].

Tensor Train layer

Let $\mathbf{x}$ be the input, $\mathbf{y}$ be the output:

$$\mathbf{y} = \mathbf{W}\mathbf{x}.$$



The matrix $\mathbf{W}$ is represented in the TT-format:

$$y(i_1, \dots, i_d) = \sum_{j_1, \dots, j_d} G_1[i_1, j_1] \dots G_d[i_d, j_d] x(j_1, \dots, j_d).$$

Parameters: TT-cores $\{\mathbf{G}_k\}_{k=1}^d$.

Details: **Tensorizing Neural Networks [Novikov et al., NIPS 2015]**.

Conclusions

- TT-decomposition and corresponding algorithms are a good way to work with huge tensors.
- Memory and complexity depend on d linearly $\Rightarrow$ no curse of dimensionality.
- TT-format is efficient only if the TT-ranks are small. This is the case in many applications.
- Code is available:
  - Python: `https://github.com/oseledets/ttpy`
  - MATLAB: `https://github.com/oseledets/TT-Toolbox`

Thanks for attention!