Tensor networks and deep learning

I. Oseledets, A. Cichocki

Skoltech, Moscow

26 July 2017

Skoltech
Сколковский институт науки и технологий

What is a tensor

Tensor is d-dimensional array:

$$A(i_1, \ldots, i_d)$$

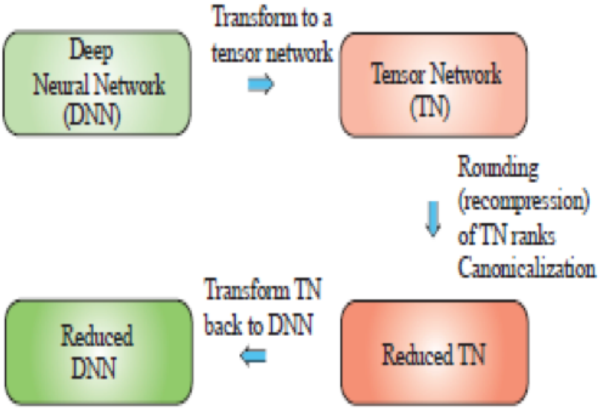
Why tensors

Many objects in machine learning can be treated as tensors:

- Data cubes (RGB images, videos, different shapes/orientations)
- Any multivariate function over tensor-product domain can be treated as a tensor
- Weight matrices can be treated as tensors, both in Conv-layers and fully-connected layers

Using tensor decompositions we can compress data!

Compression of neural networks



Compression of conv-layers

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

In a generalized convolution the kernel tensor is 4D $(d \times d \times S \times T)$ (spatial, input, output).

If we construct rank-R CP-decomposition, that amounts to having two layers of smaller total complexity, than the full layer.

The idea: use TensorLab (best MATLAB code for CP-decomposition) to initialize these two layers, and then fine-tune

Result: 8.5x speedup with 1% accuracy drop.

Compression of FC-layer

Novikov, Alexander, et al. "Tensorizing neural networks." Advances in Neural Information Processing Systems. 2015.

Use tensor-structured representation, up to 1000x compression of a fully-connected layer.

Tensor RNN

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 regularization)

Idea of tensorization

We can find tensors even in simple object!

Quantized Tensor Train format:

Take a function $f(x) = \sin x$, discretize in on $2 \times ... \times 2$ grid

Reshape into a d-dimensional tensor.

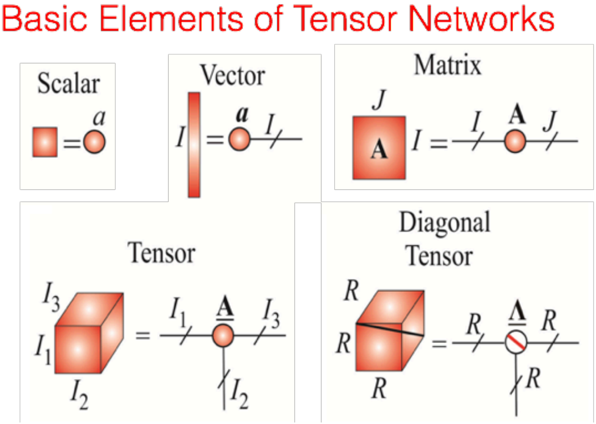
Gives $\log N$ complexity to represent classes of functions.

Connection between TN and Deep Learning

Recent work by Cohen, Shahua et. al.

Shows that tensor decompositions are neural networks with product pooling

Tensor notation



Simplest tensor network

The simplest tensor network is matrix factorization:

$$A = UV^\top.$$

Why matrix factorization is great

$$A \approx UV^\top$$

- Best factorization by SVD
- Riemmanian manifold structure
- Nice convex relaxation (nuclear norm)
- Cross approximation / skeleton decomposition

Cross approximation / skeleton decomposition

One of underestimated matrix facts:

If a matrix is rank r, it can be represented as

$$A = C\widehat{A}^{-1}R,$$

where C are some r columns of A, R are some rows of A, $\widehat{A}$ is a submatrix on the intersection.

Skoltech
Сколковский институт науки и технологий

Maximum-volume principle

Goreinov, Tyrtyshnikov, 2001 have shown:

If $\widehat{A}$ has maximal volume, then

$$\|A - A_{skel}\|_C \leq (r+1)\sigma_{r+1}.$$

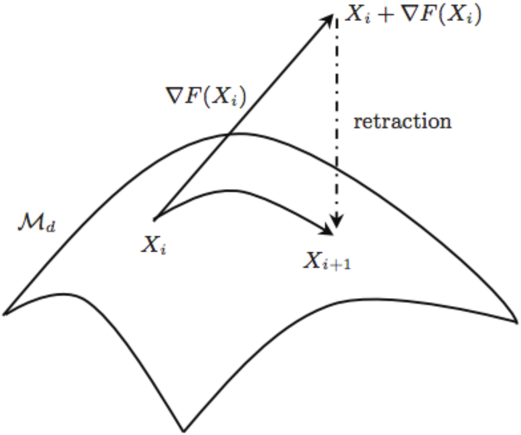
Way to compare submatrices!

Riemannian framework

Low-rank matrices form a manifold

Standard: $F(X) = F(UV^\top) \to min$

Riemmanian:



Riemannian word embedding

Example: Riemannian Optimization for Skip-Gram Negative Sampling A Fonarev, O Hrinchuk, G Gusev, P Serdyukov arXiv:1704.08059, ACL 2017.

We treated SGNS as implicit matrix factorization and solved in using Riemannian optimization.

Tensor factorization

Tensor factorization: we want numerical tools of the same quality

Skoltech

Сколковский институт науки и технологий

Classical attempt

Matrix case:

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

CP-decomposition:

$$A(i,j,k) = \sum_{\alpha=1}^{r} U(i,\alpha)V(j,\alpha)W(k,\alpha)$$

Tucker decomposition:

$$A(i,j,k) = \sum_{\alpha,\beta,\gamma=1}^{r} G(\alpha,\beta,\gamma)U(i,\alpha)V(j,\beta)W(k,\gamma)$$

CP-decomposition has bad properties!

- Best rank-r approximation may not exist
- Algorithms may converge very slowly (swamp behaviour)
- No finite-step completion procedure.

Example where CP decomposition is not known

Consider a $9 \times 9 \times 9$ tensor A with slices

$$A_i = E_i \otimes I_3, \quad i = 1, \dots, 9,$$

and E_3 has only one identity element.

It is known that CP-rank of A is ≤ 23 and ≥ 20.

Example where CP decomposition does not exist

Consider

$$T = a \otimes b \otimes \ldots \otimes b + \ldots + b \otimes \ldots \otimes a.$$

Then,

$$P(t) = \otimes_{k=1}^{d}(b+ta), \quad P'(0) = T = \frac{P(h) - P(0)}{h} + \mathcal{O}(h).$$

Can be approximated with rank-2 with any accuracy, but no exact decomposition of rank less than d exist!

Our idea

Our idea was to build tensor decompositions using well-established matrix tools.

Reshaping tensor into matrix

Let reshape an $n \times n \times \ldots \times n$ tensor into a $n^{d/2} \times n^{d/2}$ matrix A:

$$\mathbb{A}(\mathcal{I}, \mathcal{J}) = A(i_1 \ldots i_k; i_{k+1} \ldots i_d)$$

and compute low-rank factorization of $\mathbb{A}$:

$$\mathbb{A}(\mathcal{I}, \mathcal{J}) \approx \sum_{\alpha=1}^{r} U(\mathcal{I}, \alpha) V(\mathcal{J}, \alpha).$$

Recursion

If we do it recursively, we get $r^{log\, d}$ complexity

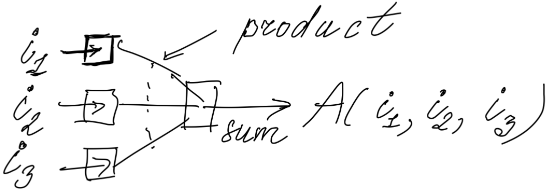
If we do it smart, we get dnr^3 complexity:

- Tree-Tucker format (Oseledets, Tyrtyshnikov, 2009)
- H-Tucker format (Hackbusch, Kuhn, Grasedyck, 2011)
- Simple but powerful version: Tensor-train format (Oseledets, 2009)

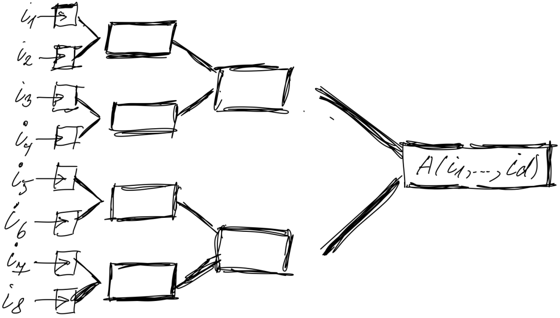
Canonical format and shallow network

N. Cohen, A. Shashua et. al provided an interpretation of the canonical format as a shallow neural network with a product pooling

$$A(i_1, \dots, i_d) \approx \sum_{\alpha=1}^{r} U_1(i_1, \alpha) U_2(i_2, \alpha) \dots U_d(i_d, \alpha).$$



H-Tucker as a deep neural network with product pooling



$$u_\gamma = \sum_{\alpha\beta} M_{\alpha\beta\gamma} v_\alpha w_\beta$$

Tensor-train

TT-decomposition is defined as

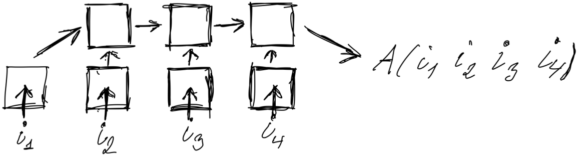
$$A(i_1, \ldots, i_d) = G_1(i_1) \ldots G_d(i_d),$$

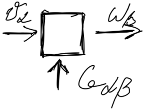
$G_k(i_k)$ is $r_{k-1} \times r_k$, $r_0 = r_d$.

Known for a long time as matrix product state in solid state physics.

Tensor-train as recurrent neural network

$$A(i_1, \dots, i_d) = G_1(i_1) \dots G_d(i_d),$$





$$w_\beta = \sum_\alpha v_\alpha G_{\alpha\beta}$$

Properties of the TT-format

- TT-ranks are ranks of matrix unfoldings
- We can do basic linear algebra
- We can do rounding
- We can recover a low-rank tensor from $\mathcal{O}(dnr^2)$ elements
- Good for rank-constrained optimization
- There are classes of problems where $r_k \sim log^s \varepsilon^{-1}$
- We have MATLAB, Python and Tensorflow toolboxes!

TT-ranks are matrix ranks

Define unfoldings:

$A_k = A(i_1 \ldots i_k; i_{k+1} \ldots i_d)$, $n^k \times n^{d-k}$ matrix

TT-ranks are matrix ranks

Define unfoldings:

$A_k = A(i_1 \ldots i_k; i_{k+1} \ldots i_d)$, $n^k \times n^{d-k}$ matrix   Theorem: there exists a TT-decomposition with TT-ranks

$$r_k = \mathit{rank}\, A_k$$

TT-ranks are matrix ranks

The proof is constructive and gives the TT-SVD algorithm!

TT-ranks are matrix ranks

No exact ranks in practice – stability estimate!

TT-ranks are matrix ranks

Physical meaning of ranks of unfoldings is entanglement: we split the system into two halves, and if rank is 1, they are independent.

Approximation theorem

If $A_k = R_k + E_k$, $||E_k|| = \varepsilon_k$

$$||\mathrm{A} - \mathrm{TT}||_F \leq \sqrt{\sum_{k=1}^{d-1} \varepsilon_k^2}.$$

TT-SVD

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 $r_3 \times n_4$

Fast and trivial linear algebra

Addition, Hadamard product, scalar product, convolution

All scale linear in d

Fast and trivial linear algebra

$$C(i_1, \ldots, i_d) = A(i_1, \ldots, i_d) B(i_1, \ldots, i_d)$$

$$C_k(i_k) = A_k(i_k) \otimes B_k(i_k),$$

ranks are multiplied

Tensor rounding

$\mathbb{A}$ is in the TT-format with suboptimal ranks.
How to reapproximate?

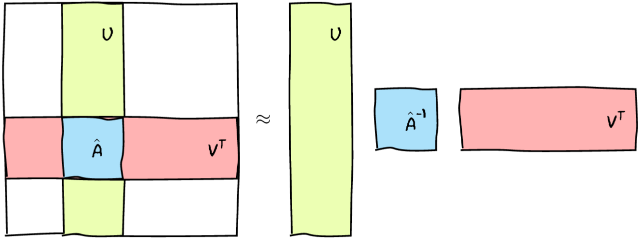
Tensor rounding

ε-rounding can be done in $\mathcal{O}(dnr^3)$ operations

Cross approximation

Recall the cross approximation

Rank-r matrix can be recovered from r columns and r rows



TT-cross approximation

Tensor with TT-ranks $r_k \leq r$ can be recovered from $\mathcal{O}(dnr^2)$ elements.

There are effective algorithms for computing those points in active learning fashion.

They are based on the computation of maximum-volume submatrices.

Making everything a tensor: the QTT

Let $f(x)$ be a univariate function (say, $f(x) = \mathit{sin}\, x$).

Let v be a vector of values on a uniform grid with 2^d points.

Transform v into a $2 \times 2 \times \ldots \times 2$ d-dimensional tensor.

Compute TT-decomposition of it!

And this is the QTT-format

Making everything a tensor: the QTT

If $f(x)$ is such that

$$f(x+y) = \sum_{\alpha=1}^{r} u_{\alpha}(x) v_{\alpha}(y),$$

then QTT-ranks are bounded by r

Corollary:

- $f(x) = exp(\lambda x)$
- $f(x) = sin(\alpha x + \beta)$
- $f(x)$ is a polynomial
- $f(x)$ is a rational function

Optimization with low-rank constraints

Tensors can be given implicitly as a solution of a certain optimization

$$F(X) \to \textit{min}, \quad r_k \leq r.$$

The set of low-rank tensors is non-convex, but has efficient *Riemannian structure* and many fabulous unstudied geometrical properties.

Desingularization

Desingularization of low-rank matrix manifolds
(V. Khrulkov, I. Oseledets).

The set of matrices of rank smaller than r is not a manifold (any matrix of smaller rank is a singular point).

Desingularization of matrix varieties

Solution: consider pairs (A, Y) such that

$$AY = 0, \quad Y^\top Y = I, \quad Y \in \mathbb{R}^{m \times (n-r)}.$$

. Theorem. Pairs (A, Y) form a smooth manifold.

We can use pain-free second-order methods to optimize with low-rank constraints.

Software

- http://github.com/oseledets/TT-Toolbox – MATLAB
- http://github.com/oseledets/ttpy – Python
- https://github.com/Bihaqo/t3f – Tensor Train in Tensorflow (Alexander Novikov)

Application of tensors

- High-dimensional, smooth functions
- Computational chemistry (electronic and molecular computations, spin systems)
- Parametric PDEs, high-dimensional uncertainty quantification
- Scale-separated multiscale problems
- Recommender systems
- Compression of convolutional layers in deep neural networks
- TensorNet (Novikov et. al) – very compact dense layers

Type of problems we can solve

- Active tensor learning by the cross method
- Solution of high-dimensional linear systems: $A(X) = F$
- Solution of high-dimensional eigenvalue problems $A(X) = \lambda X$
- Solution of high-dimensional time-dependent problems $\frac{dA}{dt} = F(A)$ (very efficient integrator).

Software

We have implemented tensor-train functionality in Tensorflow.

t3f

A library for working with Tensor Train on TensorFlow.

`https://github.com/Bihaqo/t3f`

- GPU support;
- Easy to combine with neural networks;
- Riemannian optimization support

Exponential machines

A. Novikov, M. Trofimov, I. Oseledets, Exponential Machines

Idea: use as features x_1, $x_1 x_2$, ...

There are 2^d coefficients, thus we can put low-rank constraint, and the model is

$$f(x_1, \ldots, x_d) \approx f_1(x_1, i_1) \ldots f_d(x_d, i_d) W(i_1, \ldots, i_d),$$

and then we put low-rank constraint on W.

Riemannian gradient modelling.

Model-based tensor reinforcement learning

Alex Gorodetsky, PhD dissertation, MIT (2017): tensor-train for reinforcement learning

Key components:

- "Physical" state space (like x, y, z, v_x, v_y, v_z)
- Model: $\frac{dx}{dt} = f(x, u) + \delta \frac{dW}{dt}$

As a result, Hamilton-Jacobi-Bellman equation for the optimal policy is solved using the cross method (note fundamental differences to DNN-based Q-learning!)

Comments and open problems

- Tensor decompositions are good for regression of smooth functions (neural networks are not!)
- An important question: why deep learning works so well for classification (the number of data points is much smaller, than the number of parameters)
- Can we combine the best of those approaches?

Some insights

- Tensor networks and convolutional aritmetic circuits are the same!
- Network architecture reflects "correlations" between subsystems.
- Main question: can we (and need we?) use matrix factorizations to get better algorithms for feed-forward networks?

Megagrant

2017-2019: Megagrant under the guidance of Prof. Cichocki @Skoltech (deeptensor.github.io), "Deep leaning and tensor networks".

- Victor Lempitsky (deep learning and computer vision)
- Dmitry Vetrov (deep learning and Bayesian methods)
- Ivan Oseledets (tensors)

Two monographs in Foundations and Trends in Machine Learning with basic introduction to the field.