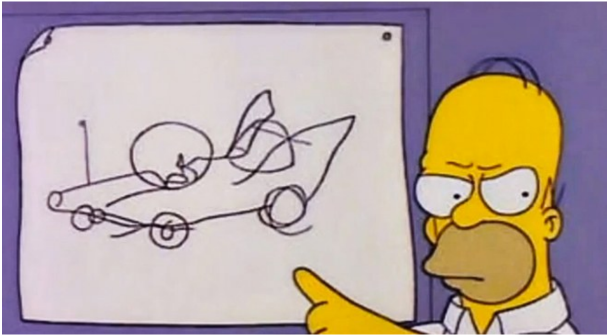
Recent advances on fast nonnegative tensor factorization

Jeremy E. Cohen

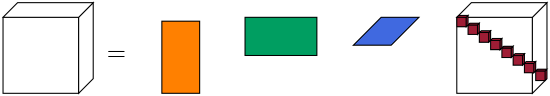
IRISA, INRIA, CNRS, University of Rennes, France

Tensor and AI workshop, Santa Fe, 18 september 2019

Goal: give an overview of research directions in fast NTF computation.



If any work of importance is not mentioned, please ask questions or talk to me!



$$
\begin{array}{ccccccccc}
\mathcal{T} & = & (A & \otimes & B & \otimes & C) & & \mathcal{I}_r \\
\mathcal{T} & = & [\![A & ; & B & ; & C]\!] & & \\
\mathcal{T} & = & A & \times_1 & B & \times_2 & C & \times_3 & \mathcal{I}_r
\end{array}
$$

Nonnegativity

$\mathcal{T} \geq 0$; $A \geq 0$, $B \geq 0$, $C \geq 0$.

Approximate NTF

Fix r, given $\mathcal{T}$, solve

$$\underset{A\geq 0, B\geq 0, C\geq 0}{\operatorname{argmin}} \|\mathcal{T} - (A \otimes B \otimes C)\mathcal{I}_r\|_F^2 \tag{1}$$

Well-posed problem, often essentially unique solution [Comon, Qi, Lim 2014]

Challenges

- Extremely large, sparse tensors
- Partial observations, sequential
- Diversity of applications and specializations
- Ill-conditioning
- Low-latency processing of average-sized tensors
- ...

Challenges

- Extremely large, sparse tensors
- Partial observations, sequential
- Diversity of applications and specializations
- Ill-conditioning
- Low-latency processing of average-sized tensors
- ...

Background: some algorithms for NTF

All-at-once

- Gradient + [Prox / Fast / *Stochastic* / Conjugate]
- ADMM
- Gauss-Newton with CG
- Levenberg Marquardt

nonnegativity imposed by interior point methods, squaring or active set.

X ADMM < AOADMM, PG < APG

X Sometimes slower than BCD

O (Second order) Very efficient near optimum

Block coordinate (alternating)

- Alternating proximal gradient
- Alternating nonnegative least squares (ANLS)
- HALS
- Multiplicative updates
- AOADMM

nonnegativity imposed mostly by proximal step.

X Sometimes slower near optimum than all-at-once

O Convex optimization tools

O Fast in practice

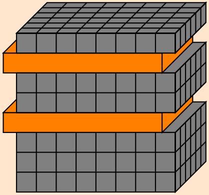
HPC

Not my expertise. . .

- n-mode product
- NNLS
- ??

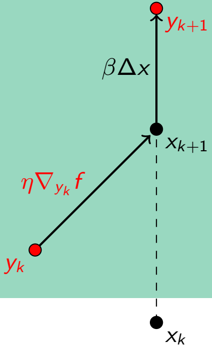
Sampling and Randomization

- Compression
- Sketching (NN ?)
- Subtensor sampling
- Fiber sampling
- Element-wise sampling



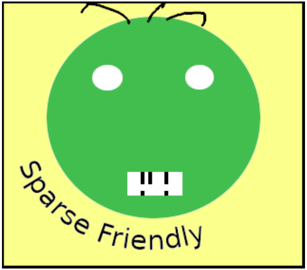
Acceleration

- Adagrad
- Momentum
- Quantification
- Extrapolation



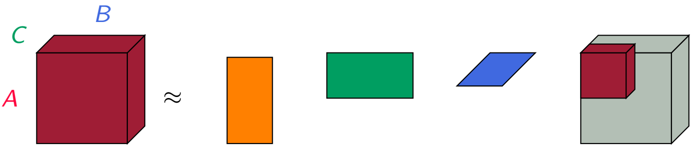
Core ideas

- Use parallel processing
- Minimal memory cost
- Utilize sparsity



Some works I know of:

1. Efficient N-mode product [Li et. al. 2015]
2. Parallel Nonnegative Least Squares for NTF [Ballard et. al., 2018]
3. Many more I do not know.



$$\mathcal{T} \approx (U \otimes V \otimes W)\,\mathcal{G}$$

Method 1: Tucker compression preprocessing [C. et. al. 2015]

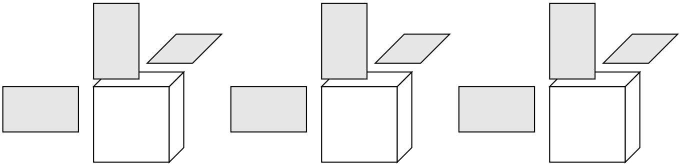
$$\underset{A_c, B_c, C_c}{\operatorname{argmin}} \|\mathcal{G} - (A_c \otimes B_c \otimes C_c)\mathcal{I}_r\|_F^2 \;\text{ s.t. } UA_c \geq 0, VB_c \geq 0, WC_c \geq 0 \tag{2}$$

Method 2: Account for (Tucker) structure [Vervliet et. al. 2019]

$$\underset{A,B,C}{\operatorname{argmin}} \|(U \otimes V \otimes W)\mathcal{G} - (A \otimes B \otimes C)\mathcal{I}_r\|_F^2 \;\text{ s.t. } A \geq 0, B \geq 0, C \geq 0 \tag{3}$$

Idea: random projections

1. [CPD-(NTF?)] PARACOMP [Sidiropoulos, Papalexakis et al 2014]



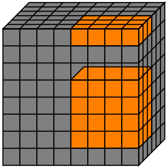
2. [Tucker] Tensor sketching with DRM [Sun et. al. 2019]
3. [NN Tucker] TENSORSKETCH [e.g. Anandkumar 2015] [Malik, Becker 2018] (Fast implementation with FFT, COUNTSKETCH)

Not really adapted to NTF? Or interesting perspectives?

Sampling and Randomization 3: subtensor / slices sampling

A lot of existing and ongoing work

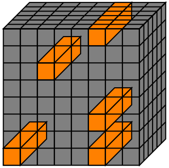
1. [CPD-NTF] Subtensor sampling [Papalexakis et. al. 2014] [Vervliet et. al. 2016]



Makes (stochastic) gradient steps

- cheap
- memory-efficient
- partial (only some parameters are updated)

2. [CPD-NTF] Fibers sampling [Battaglino et. al., 2018] [Fu et. al. 2019?]

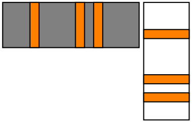


Makes MTTKRP

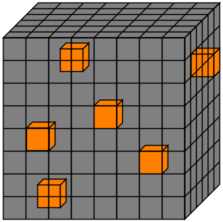
- cheaper
- memory-efficient

easy sampling with kr product!

$T_{[3]}(A \odot B)^T$



Sampling and Randomization 4: element-wise sampling



1. [Tucker] Sampling once and reweighting: MACH [Tsurakakis, 2009]
2. Stochastic gradient? [Kolda et. al. ??]

Ongoing research topic! Naive stochastic gradient is bad?

Some observations

- Tensor structure makes us creative
- Room for nonnegative adaptations
- Strong sampling / optimization / implementation ties

Proximal gradient-based method $\Longrightarrow$ Acceleration!!

- Extrapolation of the iterates (Nesterov, Andersen...)
- Stepsize adaptation
- Gradient momentum
- Gradient quantification / noise
- ...

e.g.: Adam and (N)TF in [Fu 2019?][Kolda?][Some random online blog]

Question

Is this really research material for ~~us~~ me?

Cons

- Straightforward in Pytorch
- Mostly development
- Super incremental
- **Need consistent tests**

Pros

- Nice interactions
- Convergence proofs
- Niche effect?
- **New accelerations!**

Acceleration of first order methods

Proximal gradient-based method $\implies$ Acceleration!!

- Extrapolation of the iterates (Nesterov, Andersen...)
- Stepsize adaptation
- Gradient momentum
- Gradient quantification / noise
- ...

e.g.: Adam and (N)TF in [Fu 2019?][Kolda?][Some random online blog]

Question

Is this really research material for ~~us~~ me?

Cons

- Straightforward in Pytorch
- Mostly development
- Super incremental
- **Need consistent tests**

Pros

- Nice interactions
- Convergence proofs
- Niche effect?
- **New accelerations!**

Extrapolation for NTF

All-at-once optimization:

Mostly straightforward, except for second order methods(?)

BCD:

Extrapolate within each block update!

e.g.: Alternatively solve for A, B, C

$$\underset{A \geq 0}{\operatorname{argmin}} \| T_{[1]} - A(B \odot C)^T \|_F^2 \quad \text{(Matrix NNLS)} \tag{4}$$

solved with extrapolated ADMM [Liavas et. al. 2018].

Another ref with HPC acceleration instead [Smith et. al. 2017]

Time for our contribution:
extrapolation between each block in BCD



Extrapolation: Heuristic Extrapolation with Restart (HER)

The HER algorithm

- initialize U, V, W; $Uy = U, Vy = V, Wy = W$
- loop until convergence:
  1. $U_{old} = U, V_{old} = V; W_{old} = W$
  2. Update β with heuristic (next slide)

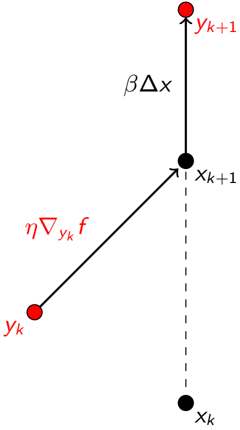
  3. Update U — e.g. using NNLS($\mathcal{T}, Vy, Wy$)
  4. Extrapolate $Uy = U + \beta(U - U_{old})$

  5. Update V — e.g. using NNLS($\mathcal{T}, Uy, Wy$)
  6. Extrapolate $Vy = V + \beta(V - V_{old})$

  7. Update W — e.g. using NNLS($\mathcal{T}, Uy, Vy$)
  8. Extrapolate $Wy = W + \beta(W - W_{old})$
- if cost function increases, restart $Uy = U, Vy = V, Wy = W$

<u>What is "new":</u> Extrapolation between blocks of BCD! [Bro 1998]
<u>What is common:</u> Extrapolation within each block.

Extrapolation: HER continued



At each iteration,

1. if error has decreased, increase β up to a threshold β_{max}.
2. if error has increased, decrease β and β_{max}.

In any case, $\beta \in]0, \beta_{max}]$ with $\beta_{max} \leq 1$.

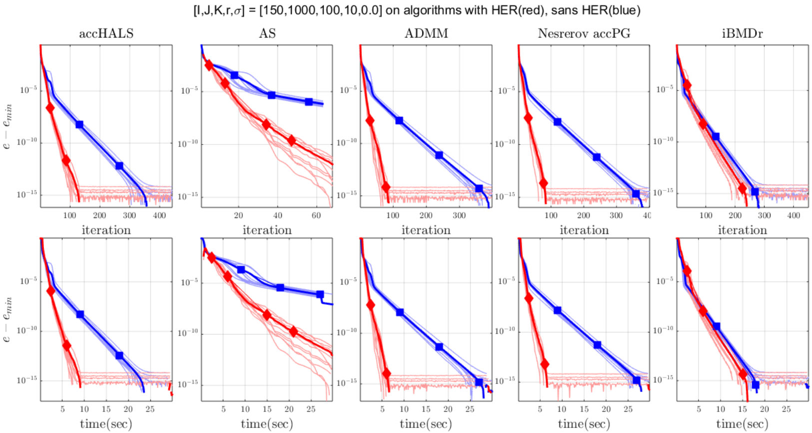


Figure: Acceleration by HER (red) vs BCD (blue) with various update strategies

Conclusion

Fast NTF

- Many approaches in the era of Machine Learning
- Cross-disciplinary = interactions!!
- **Need comparisons!! Need benchmark data**
  (sparse/dense, average/huge, various applications)

Fast NTF		
HPC	Sampling	Acceleration

Proposed algorithm HER

- Easy to understand, hard to study
- Plug-and-play
- Promising results [Ang, C., Gillis 2019][A., Hien, C., G. in prep.] for NTF
- Can interact with most discussed methods for fast NTF!!

A word from my co-author

A fourth ingredient: pre-conditionning?

Conclusion

Fast NTF

- Many approaches in the era of Machine Learning
- Cross-disciplinary = interactions!!
- **Need comparisons!! Need benchmark data**
  (sparse/dense, average/huge, various applications)

Fast NTF		
HPC	Sampling	Acceleration

Proposed algorithm HER

- Easy to understand, hard to study
- Plug-and-play
- Promising results [Ang, C., Gillis 2019][A., Hien, C., G. in prep.] for NTF
- Can interact with most discussed methods for fast NTF!!

A word from my co-author

A fourth ingredient: pre-conditionning?