BlockTensorDecompositions.jl: A Unified Constrained Tensor Decomposition Julia Package

Nicholas J. E. Richardson
Department of Mathematics

Noah Marusenko
Department of Computer Science

Michael P. Friedlander
Departments of Mathematics and Computer Science

Table of contents

1 Introduction

- Tenors are useful in many applications
- Need tools for fast and efficient decompositions

For the scientific user, it would be most useful for there to be a single piece of software that can take as input, any reasonable type of factorization model, and constraints on the individual factors, and produce a factorization without worrying the user about the details of what rank to select, how the constraints should be enforced, and how to optimize for performance. Of course a knowledgeable user may still want the ability to tweak the convergence criteria used, the loss function optimized, or what statistics to record each iteration. These are the core specification for BlockTensorDecompositions.jl.

1.1 Related tools

- Packages within Julia

- Other languages
- Hint at why I developed this

Beyond the external usefulness already mentioned, this package offers a playground for fair comparisons of different parameters and options for performing tensor factorizations across various decomposition models. There exist packages for working with tensors in languages like Python (TensorFlow [1], PyTorch [2], and TensorLy [3]), MATLAB (Tensor Toolbox [4]), R (rTensor [5]), and Julia (TensorKit.jl [6], Tullio.jl [7], OMEinsum.jl [8], and TensorDecompositions.jl [9]). But they only provide a groundwork for basic manipulation of tensors and the most common tensor decomposition models and algorithms, and are not equipped to handle arbitrary user defined constraints and factorization models.

Some progress towards building a unified framework has been made [10–12]. But these approaches don't operate on the high dimensional tensor data natively and rely on matricizations of the problem, or only consider nonnegative constraints. They also don't provide an all-in-one package for executing their frameworks.

1.2 Contributions

- Fast and flexible tensor decomposition package
- Framework for creating and performing custom
  - tensor decompositions
  - constrained factorization (the what)
  - iterative updates (the how)
- Implement new "tricks"
  - a (Lipschitz) matrix step size for efficient sub-block updates
  - multi-scaled factorization when tensor entries are discretizations of a continuous function
  - partial projection and rescaling to enforce linear constraints (rather than Euclidean projection)
  - ?? rank detection ??

2 Tensor Decompositions

- the math section of the paper

2.1 Notation

- tensor notation, use MATLAB notation for indexing so subscripts can be used for a sequence of tensors

2.2 Common Decompositions

- Extensions of PCA/ICA/NMF to higher dimensions
- talk about the most popular Tucker, Tucker-n, CP
- other decompositions
  - high order SVD (see Kolda and Bader)
  - HOSVD (see Kolda, Shifted power method for computing tensor eigenpairs)

2.3 Tensor rank

- tensor rank
- constrained rank (nonnegative etc.)

3 Computing Decompositions

- Given a data tensor and a model, how do we fit the model?

3.1 Optimization Problem

- Least squares (can use KL, 1 norm, etc.)

3.2 Base algorithm

- Use Block Coordinate Descent / Alternating Proximal Descent
  - do *not* use alternating least squares (slower for unconstrained problems, no closed form update for general constrained problems)

4 Techniques for speeding up convergences

- As stated, algorithm works
- But can be slow, especially for constrained or large problems

4.1 Sub-block Descent

- Use smaller blocks, but descent in parallel (sub-blocks don't wait for other sub-blocks)
- Can perform this efficiently with a "matrix step-size"

4.2 Momentum

- This one is standard
- Use something similar to [10]
- This is compatible with sub-block descent with appropriately defined matrix operations

4.3 Partial Projection and Rescaling

- for bounded linear constraints
  - first project
  - then rescale to enforce linear constraints
- faster to execute then a projection
- often does not loose progress because of the rescaling (decomposition dependent)

4.4 Multi-scale

- use a coarse discretization along continuous dimensions
- factorize
- linearly interpolate decomposition to warm start larger decompositions

5 Conclusion

- all-in-one package
- provide a playground to invent new decompositions
- like auto-diff for factorizations

Bibliography

[1] Martín Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems,” 2015. https://www.tensorflow.org/

[2] J. Ansel *et al.*, “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation,” in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, Apr. 2024, vol. 2, pp. 929–947. doi: 10.1145/3620665.3640366.

[3] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, “TensorLy: Tensor Learning in Python,” *Journal of Machine Learning Research*, vol. 20, no. 26, pp. 1–6, 2019, Accessed: Aug. 16, 2024. [Online]. Available: http://jmlr.org/papers/v20/18-277.html

[4] B. W. Bader and T. G. Kolda, “Tensor Toolbox for MATLAB.” Sep. 2023.

[5] J. Li, J. Bien, and M. T. Wells, “rTensor: An R Package for Multidimensional Array (Tensor) Unfolding, Multiplication, and Decomposition,” *Journal of Statistical Software*, vol. 87, pp. 1–31, Nov. 2018, doi: 10.18637/jss.v087.i10.

[6] Jutho, “Jutho/TensorKit.jl,” Aug. 2024. https://github.com/Jutho/TensorKit.jl (accessed Aug. 15, 2024).

[7] M. Abbott *et al.*, “mcabbott/Tullio.jl: v0.3.7,” Oct. 2023. https://doi.org/10.5281/zenodo.10035615

[8] A. Peter, “under-Peter/OMEinsum.jl,” Aug. 2024. https://github.com/under-Peter/OMEinsum.jl (accessed Aug. 16, 2024).

[9] Y.-J. Wu, “yunjhongwu/TensorDecompositions.jl,” Feb. 2024. https://github.com/yunjhongwu/TensorDecompositions.jl (accessed Aug. 16, 2024).

[10] Y. Xu and W. Yin, “A Block Coordinate Descent Method for Regularized Multiconvex Optimization with Applications to Nonnegative Tensor Factorization and Completion,” *SIAM Journal on Imaging Sciences*, vol. 6, no. 3, pp. 1758–1789, Jan. 2013, doi: 10.1137/120887795.

[11] J. Kim, Y. He, and H. Park, “Algorithms for nonnegative matrix and tensor factorizations: a unified view based on block coordinate descent framework,” *Journal of Global Optimization*, vol. 58, no. 2, pp. 285–319, Feb. 2014, doi: 10.1007/s10898-013-0035-4.

[12] Z. Yang and E. Oja, “Unified Development of Multiplicative Algorithms for Linear and Quadratic Nonnegative Matrix Factorization,” *IEEE Transactions on Neural Networks*, vol. 22, no. 12, pp. 1878–1891, Dec. 2011, doi: 10.1109/TNN.2011.2170094.