

TensorLy: Tensor Learning in Python

Jean Kossaifi¹

Yannis Panagakis^{1,2}

Anima Anandkumar^{3,4}

Maja Pantic¹

¹*Imperial College London*

³ *NVIDIA*

JEAN.KOSSAIFI@IMPERIAL.AC.UK

I.PANAGAKIS@IMPERIAL.AC.UK

ANIMA@CALTECH.EDU

M.PANTIC@IMPERIAL.AC.UK

²*Middlesex University*

⁴ *California Institute of Technology*

Editor: Alexandre Gramfort

Abstract

Tensors are higher-order extensions of matrices. While matrix methods form the cornerstone of traditional machine learning and data analysis, tensor methods have been gaining increasing traction. However, software support for tensor operations is not on the same footing. In order to bridge this gap, we have developed *TensorLy*, a Python library that provides a high-level API for tensor methods and deep tensorized neural networks. TensorLy aims to follow the same standards adopted by the main projects of the Python scientific community, and to seamlessly integrate with them. Its BSD license makes it suitable for both academic and commercial applications. TensorLy's backend system allows users to perform computations with several libraries such as NumPy or PyTorch to name but a few. They can be scaled on multiple CPU or GPU machines. In addition, using the deep-learning frameworks as backend allows to easily design and train deep tensorized neural networks. TensorLy is available at <https://github.com/tensorly/tensorly>

1. Introduction

Tensors are higher-order extensions of matrices. While matrices are indexed by two indices (and hence, second order tensors), tensors can be indexed by an arbitrary number of indices. Tensors have a rich history, stretching almost a century, and have been used in diverse fields such as psychometrics, quantum systems, signal processing etc. Only recently have tensors been employed in machine learning and data analytics. This can be attributed to the availability of large-scale multi-dimensional and multi-modal datasets. Tensors form the natural framework to encode and operate on such data structures.

For multi-dimensional datasets, there are natural extensions of traditional dimensionality-reduction methods, such as principal component analysis (PCA), to higher dimensions. These involve tensor decompositions and have been applied in a number of areas, including the theoretical analysis of deep neural nets (Cohen et al., 2015). Another line of work aims to operate on higher-order moments of the data distribution, i.e., beyond pairwise correlations. This is shown to be fruitful for learning a wide range of probabilistic latent-variable models such as Gaussian mixtures, Independent Component Analysis, topic models etc (Anandkumar et al., 2014). More recently, deep tensorized neural networks have been explored for a variety of applications. They extend linear-algebraic operations in various layers to tensor algebraic operations, such as tensor contraction and tensor regression, in convolutional

architectures (Novikov et al., 2015; Kossaifi et al., 2017), generative adversarial networks (Cao and Zhao, 2017) and sequence models (Yu et al., 2017).

Interested readers are referred to several surveys on this topic. Some works focus on the basics of multi-linear (tensor) algebra and different types of tensor decompositions, (Kolda and Bader, 2009; Sidiropoulos et al., 2016). Others focus on algorithmic advances, (Cichocki et al., 2009; Lu et al., 2011; Grasedyck et al., 2013; Cichocki et al., 2015; Papalexakis et al., 2016). Recent surveys focus on their applications (Acar and Yener, 2009) and uses in learning probabilistic models (Janzamin et al., 2019).

Thus, tensor methods can have a profound impact on data analytics and machine learning with clear theoretical, algorithmic, and practical advantages over their matrix counterparts. However, as opposed to matrix methods, tensor methods have not yet been widely adopted by data scientists. This can be mainly attributed to the lack of available software libraries for tensor operations and decompositions, in programming languages that data scientists and practitioners are familiar with (e.g., Python, Java, Scala, etc).

Even though some tensor libraries exist, they are implemented in non-free platforms (e.g., MATLAB’s TensorToolbox, (Bader and Kolda, 2015) and TensorLab, (Vervliet et al., 2016)) or in low-level languages like C++ (e.g., TH++). Python is emerging as a language of choice for machine learning, as witnessed with the success of scikit-learn (Pedregosa et al., 2011), and is increasingly used in both academic and industrial research projects. However, there is not yet a Python library implementing tensor operations, decomposition, and learning. The existing ones (e.g., scikit-tensor) offer only limited functionalities and/or have restrictive licenses. Moreover, widely-adopted deep-learning libraries such as Tensorflow and Torch lack advanced tensor operations. For applications to data analytics, machine learning, and deep learning, there is an urgent need for well-developed and documented open-source libraries that include methods for tensor decompositions.

In this paper, to address the aforementioned need, the *TensorLy*¹ library is introduced, allowing several contributions over the existing libraries for tensor methods. In particular, TensorLy a) provides state-of-the-art tensor learning, including core tensor operations and algebra, tensor decomposition and tensor regression methods; b) has a flexible *backend* system that allows switching between NumPy, MXNet, PyTorch, TensorFlow, and CuPy to perform the computation, and to easily combine tensor methods with deep learning; c) is open source and BSD licensed; d) depends by default exclusively on NumPy and SciPy; and e) is accompanied by extensive tests and documentation.

2. *TensorLy*: functionalities and implementation

TensorLy has been developed with the goal of making tensor learning more accessible and to allow for seamless integration with the Python scientific environment. It depends by default only on Numpy (van der Walt et al., 2011) and Scipy, with a soft dependency on Matplotlib (Hunter, 2007) for plotting.

To enable algorithms to be easily run at scale, on various hardware, and to combine tensor methods with deep learning, TensorLy has a flexible backend system. This allows tensor operations to be ran using NumPy, MXNet (Chen et al., 2015), PyTorch (Paszke

1. *TensorLy*’s source code available at <https://github.com/tensorly/tensorly> and documentation at <http://tensorly.org>.

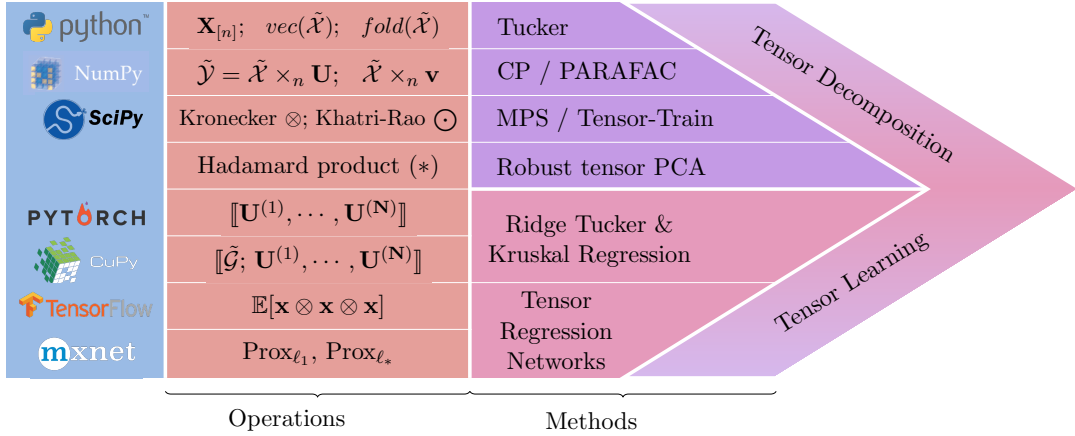


Figure 1: TensorLy builds on top of the Python ecosystem and implements Tensor Algebra *Operations*. These tensor operators are then used for higher level *Methods* such as tensor regression and decomposition, or combined with deep learning.

et al., 2017), Chainer’s CuPy (Tokui et al., 2015) or TensorFlow with eager execution (Abadi et al., 2015). **NumPy** is the standard library for numerical computation in Python. It offers high performance structures for manipulating multi-dimensional arrays. **CuPy** works as a drop-in replacement for NumPy with GPU support. **MXNet** is a flexible deep learning library with an NDAarray structure that allows to efficiently run code on CPU, GPU and multi-machines. Similarly, **PyTorch** provides a NumPy-like API with GPU acceleration and auto-grad using dynamical graphs. **TensorFlow** is an established machine-learning framework that provides an imperative approach using eager execution.

We aim at making the *API* simple and efficient, following that of scikit-learn (Buitinck et al., 2013), where possible. However, while scikit-learn works with observations (samples) represented as vectors, this library focuses on higher order arrays. *TensorLy*’s main functionalities in term of tensor operations are summarized in Fig. 2, where in the operations column the mathematical notation of Kolda and Bader (2009) is adopted. In the methods column, we summarize the implemented well-known tensor decomposition and regression methods. These include CP and Tucker decompositions, their non-negative versions, matrix-product-state (also known as tensor-train decomposition), Robust tensor PCA, (Goldfarb and Qin, 2014) and low-rank tensor (Kruskal and Tucker) Regression. Additionally, using the deep learning backends, it is easy to combine tensor methods and Deep Learning.

We emphasize code quality and ease of utilization for the end user. To that extent, both testing and documentation are an essential part of the package: all functions come with documentation and unit-tests (at the time of writing, the coverage is of 99%).

3. Performance

TensorLy has been tailored for the Python ecosystem: tensors are multi-dimensional arrays which are effectively manipulated directly by the various methods, decomposition or regression. This allows for competitive performance even though the library is implemented in a high-level, interactive language. The operations are also optimized for speed: tensor operations have been redefined when possible to allow for better performance. In particular,

we propose an efficient unfolding of tensors which differs from the traditional one (Kolda and Bader, 2009) by the ordering of the fibers.

Given a tensor, $\tilde{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$, the mode- n unfolding of $\tilde{\mathcal{X}}$ is a matrix $\mathbf{X}_{[n]} \in \mathbb{R}^{I_n \times I_M}$, and is defined by the mapping from element (i_1, i_2, \dots, i_N) to (i_n, j) ,

$$\text{with } j = \sum_{\substack{k=1, \\ k \neq n}}^N i_k \times \prod_{\substack{m=k+1, \\ k \neq n}}^N I_m, \quad \text{and } M = \prod_{\substack{k=1, \\ k \neq n}}^N I_k$$

This formulation both achieves better performance when using C-ordering of the elements (Numpy and most Python libraries’ default), and translates into more natural properties.

3.1. Numerical test

We generated random third order tensors of size $500 \times 500 \times 500$ (125 million elements). We then compared the decomposition speed for a rank-50 CANDECOMP-PARAFAC (CP) and rank (50, 50, 50)–Tucker decomposition with TensorLy on CPU (NumPy backend) and TensorLy on GPU (MXNet, PyTorch, TensorFlow and CuPy backends), and Scikit-Tensor (*Sktensor*), Fig. 2. In all cases we fixed the number of iterations to 100 to allow for a fair comparison. The experiment was repeated 10 times, with the main bar representing the average CPU time and the tip on the bar the standard deviation of the runs. As can be observed, our library offers competitive speed, thanks to optimized formulation and implementation. Experiments were done on an Amazon Web Services p3 instance, with a NVIDIA VOLTA V100 GPU and 8 Intel Xeon E5 (Broadwell) processors.

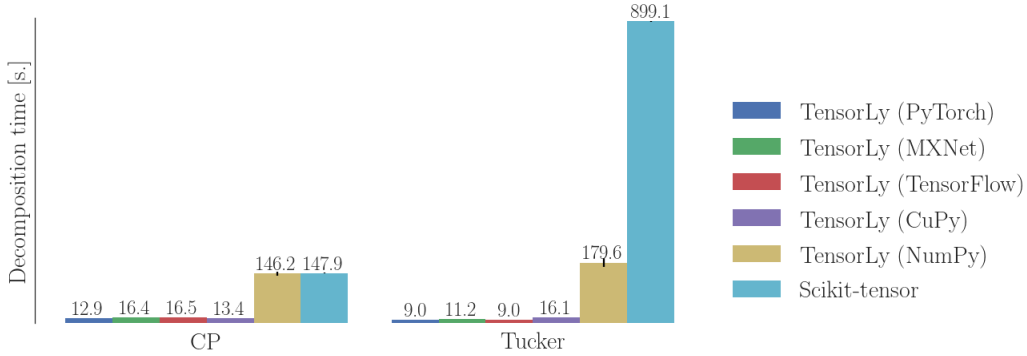


Figure 2: Speed comparison for Tucker and CANDECOMP-PARAFAC decomposition.

4. Conclusion

TensorLy makes tensor learning accessible and straightforward by offering state-of-the-art tensor methods and operations through simple and consistent interfaces, under a permissive license. It is optimized to be fast and robust, with systematic unit-tests and documentation. The library’s speed and ease of use allow for an efficient comparison of existing methods and easy implementation of new ones. Its flexible backend system allows to transparently switch between libraries and platforms and help combine tensor methods with deep learning. Going forward, we will further extend the available methods with other state-of-the-art methods and target further improvements in performance, as exemplified by Shi et al. (2016).

References

- M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org.
- E. Acar and B. Yener. Unsupervised multiway data analysis: A literature survey. *IEEE Transactions on Knowledge and Data Engineering*, 21(1):6–20, Jan 2009.
- A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *JMLR*, 15(1):2773–2832, jan 2014.
- B. W. Bader and T. G. Kolda. Matlab tensor toolbox version 2.6. Available online, February 2015.
- L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.
- X. Cao and Q. Zhao. Tensorizing generative adversarial nets. *CoRR*, abs/1710.10772, 2017.
- T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.
- A. Cichocki, R. Zdunek, A. H. Phan, and S.-I. Amari. *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-Way Data Analysis and Blind Source Separation*. John Wiley & Sons, Ltd, 2009.
- A. Cichocki, D. Mandic, L. D. Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. PHAN. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE Signal Processing Magazine*, 32(2):145–163, March 2015.
- N. Cohen, O. Sharir, and A. Shashua. On the expressive power of deep learning: A tensor analysis. *CoRR*, abs/1509.05009, 2015.
- D. Goldfarb and Z. T. Qin. Robust low-rank tensor recovery: Models and algorithms. *SIAM Journal on Matrix Analysis and Applications*, 35(1):225–253, 2014.
- L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.
- J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, 9(3): 90–95, May 2007.
- M. Janzamin, R. Ge, J. Kossaifi, and A. Anandkumar. Spectral learning on matrices and tensors. *pre-print*, 2019.
- T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM REVIEW*, 51(3): 455–500, 2009.

- J. Kossaifi, Z. C. Lipton, A. Khanna, T. Furlanello, and A. Anandkumar. Tensor regression networks. *CoRR*, abs/1707.08308, 2017.
- H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos. A survey of multilinear subspace learning for tensor data. *Pattern Recognition*, 44(7):1540 – 1551, 2011.
- A. Novikov, D. Podoprikin, A. Osokin, and D. Vetrov. Tensorizing neural networks. In *NIPS*, pages 442–450, 2015.
- E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Trans. Intell. Syst. Technol.*, 8(2):16:1–16:44, Oct. 2016.
- A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- Y. Shi, U. N. Niranjan, A. Anandkumar, and C. Cecka. Tensor contractions with extended blas kernels on cpu and gpu. In *IEEE HiPC*, pages 193–202, Dec 2016.
- N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos. tensor decomposition for signal processing and machine learning. *arXiv preprint arXiv:1607.01668*, 2016.
- S. Tokui, K. Oono, S. Hido, and J. Clayton. Chainer: a next-generation open source framework for deep learning. In *LearningSys Workshop in NIPS*, 2015.
- S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.
- N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer. Tensorlab 3.0, Mar. 2016. URL <https://www.tensorlab.net>. Available online.
- R. Yu, S. Zheng, A. Anandkumar, and Y. Yue. Long-term forecasting using tensor-train rnns. *CoRR*, abs/1711.00073, 2017.