TensorLy: Tensor Learning in Python

Jean Kossaifi
Department of Computing
Imperial College London.
jean.kossaifi@imperial.ac.uk

Yannis Panagakis
Department of Computing,
Imperial College London
i.panagakis@imperial.ac.uk

Maja Pantic
Department of Computing, Imperial College London
EEMCS, Univestiy of Twente, the Netherlands
m.pantic@imperial.ac.uk

Abstract

Tensor methods are gaining increasing traction in machine learning. However, there are scant to no resources available to perform tensor learning and decomposition in Python. To answer this need we developed *TensorLy*.

TensorLy is a state of the art general purpose library for tensor learning. Written in Python, it aims at following the same standard adopted by the main projects of the Python scientific community and fully integrating with these. It allows for fast and straightforward tensor decomposition and learning and comes with exhaustive tests, thorough documentation and minimal dependencies. It can be easily extended and its BSD licence makes it suitable for both academic and commercial applications. *TensorLy* is available at `https://github.com/tensorly/tensorly`.

1 Introduction

Tensors, or *multi-way arrays*, of order higher than two, are multi-dimensional arrays indexed by three or more indices, generalizing the notion of matrix (second order tensor). Tensors and tensor decompositions or factorizations have a rich history, stretching almost a century, but only recently i.e., roughly a decade ago, have become ubiquitous in signal processing, statistics, data analytics, and machine learning.

Indeed, psychometrics and chemometrics have historically been the application areas driving theoretical and algorithmic developments in the field. However, tensors and their decompositions were popularized by the (statisitical) signal processing and machine learning communities when they realized the power of tensor decompositions in practice. Examples of such applications include, speech, music, image, communications, biomedical, and social network signal processing and analysis, as well as clustering, dimensionality reduction, subspace, dictionary and features learning e.g., [23, 24, 30, 22, 20, 25, 14, 2].

More recently, there has been a considerable amount of research in establishing connections between tensor decompositions, the method of (high-order) moments, and compositional function spaces in order to learn latent variable models (e.g., mulitiview mixture model, Gaussian mixtures, Independent Component Analysis) [3], train deep neural networks with theoretical guarantees [13], and also theoretically analyze their impressive empirical performance [8].

The interested reader is referred to several surveys in the topic, which focus range from basics of multilinear (tensor) algebra and overview of different established tensor decompositions [17, 29], to algorithmic advances [7, 19, 10, 6, 26] and applications [1].

Based on the above discussion, tensors and their decompositions have profound impact in signal (data) analytics and machine learning with clear theoretical, algorithmic, and practical advantages

Tensor unfolding ($\mathbf{X}_{[n]}$), folding and vectorisation ($vec(\tilde{\mathcal{X}})$)
Tensor multiplication (n-mode matrix product, $\tilde{\mathcal{Y}} = \tilde{\mathcal{X}} \times_n \mathbf{U}$ and n-mode vector product $\tilde{\mathcal{X}} \times_n \mathbf{U}$)
Matrix Kronecker ($\otimes$) and Khatri-Rao ($\bigodot$) products
Hadamard product ($*$) is inherited from the standard numpy structure
Kruskal ($[\![\mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(\mathbf{N})}]\!]$) and Tucker operators ($[\![\tilde{\mathcal{G}}; \mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(\mathbf{N})}]\!]$)
Computation of Higher-Order moments ($\mathbb{E}[\mathbf{x} \otimes \mathbf{x} \otimes \mathbf{x}]$)
Proximal operators for the ℓ_1 norm and the nuclear norm

Table 1: Operations implemented

over their matrix counterparts. However, as opposed to matrix factorizations and matrix-based machine learning methods, tensor decompositions have not been widely adopted by data scientists and practitioners, yet. This can be mainly attributed to the fact that there is a lack in available libraries for tensor operations and decompositions, accessible from programming languages (e.g., Python, Java, Scala, etc) that data scientists and practitioners are familiar with. Even though some libraries exist for handling tensors, these are implemented in non-free platforms (e.g. MATLAB's TensorToobox [5] and TensorLab [21]) or in low-level languages like C++ (e.g. TH++) and the deep learning libraries e.g., Tensorflow and Torch can only suboptimally handle tensors.

Python is emerging as a language of choice for machine learning, as witnessed with the success of scikit-learn [27], and is increasingly used in both academic and industrial research projects. However, there is to date no Python library implementing tensor decomposition and learning. The exisiting ones (e.g., scikit-tensor) offer only limited algorithms (e.g., decomposition only) and/or have restrictive licenses. For applications to data analytics and machine learning, open source, well-developed and -documented libraries that include methods for tensor decompositions are urgently needed.

TensorLy[1] introduces several contributions over the existing libraries: a) it provides state of the art tensor learning including core tensor operations, tensor decomposition and tensor regression methods. b) it is open source and BSD licensed. c) it comes with extensive tests and documentation; and d) it depends exclusively on numpy and scipy.

2 *TensorLy* functionality and implementation

TensorLy has been developed with the goal to make tensor learning more accessible and to allow for seemless integration with the python scientific environment. It builds on top of two core Python libraries, Numpy [31] and Scipy [15] while having a soft-dependency on Matplotlib [12] for plotting:

Numpy The standard library for numerical computation in Python. It offers high performance structures for manipulating multi-dimensional arrays. In particular, in *TensorLy*, we leverage this convenient structure for efficient tensor operations.

Scipy Provides high performance mathematical functions, advantageously using numpy's ndarray structure.

Matplotlib Cross-compatible 2D graphics package offering high quality image and graphics generation.

The Application Programming Interface (*API*) aims at compatibility with scikit-learn [27], which is the de-facto standard library for performing classical Machine Learning, preprocessing and cross-validation. While scikit-learn is built to work with observations (samples) represented as vectors, this library focuses on higher order arrays.

TensorLy's current functionalities in term of tensor operations are summarised in Table. 2, where inside the parenthesis the mathematical notation of Kolda and Bader [17] is adopted. Furthermore, we have implemented core tensor decomposition and tensor regression methods listed in Table. 2.

TensorLy has been tailored for the Python ecosystem and the operations are optimised for speed: tensor operations have been redefined when possible to allow for better performance. In particular, we

[1] *TensorLy* is available at `https://github.com/tensorly/tensorly`

CANDECOMP-PARAFAC (CP) decomposition e.g. [17])
Non-Negative CP decomposition [28])
Tucker decomposition (Higher-Order SVD) (e.g. [17])
Non-Negative Tucker decomposition [16])
Tensor Ridge regression (Tucker and Kruskal) [11, 32, 18]

Table 2: Algorithms implemented

propose an efficient unfolding of tensors which differs from the traditional one [17] by the ordering of the columns.

Given a tensor, $\tilde{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, the mode-$n$ unfolding of $\tilde{\mathcal{X}}$ is a matrix $\mathbf{X}_{[n]} \in \mathbb{R}^{I_n, I_M}$, with $M = \prod_{\substack{k=1, \\ k \neq n}}^{N} I_k$ and is defined by the mapping from element $(i_1, i_2, \cdots, i_N)$ to (i_n, j), with $j = \sum_{\substack{k=1, \\ k \neq n}}^{N} i_k \times \prod_{m=k+1}^{N} I_m$.

Not only does this formulation achieve better performance when using C-ordering of the elements (as numpy does by default), it also translates into more natural properties. For instance, given a tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and its Tucker decomposition $[\![\tilde{\mathcal{G}}; \mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(N)}]\!]$, we can express the mode-n unfolding of $\tilde{\mathcal{X}}$ as :

$$\mathbf{X}_{[n]} = \mathbf{U}^{(n)} \mathbf{G}_{[n]} \left(\mathbf{U}^{(1)} \otimes \cdots \mathbf{U}^{(n-1)} \otimes \mathbf{U}^{(n+1)} \otimes \cdots \otimes \mathbf{U}^{(N)} \right)^T$$

Finally, we emphasize code quality and ease of utilisation for the end user. To that extent, both testing and documentation are an essential part of the package. Each function comes with its documentation and unit-tests (at the time of writing, the coverage is of 99 %).

3 Experiments

In *TensorLy*, tensors are simply numpy mutli-dimensional arrays which are passed directly to the various methods, decomposition or regression. This allows for competitive performance even though the library is implemented in a high-level, interactive language.

3.1 Tensor regression

TensorLy offers a simple interface for tensor regression with the same API as Scikit-Learn. The regressors are object that expose a *fit* method that takes as parameters the data tensor $\tilde{\mathcal{X}}$ and the corresponding array of labels $\mathbf{y}$. Given new data $\tilde{\mathcal{X}}_T$, the *predict* method returns the regressed labels $\mathbf{y}_T$.

To illustrate the effectiveness of tensor regression, we fixed the regression weights $\tilde{\mathcal{W}}$ to be a second order tensor (a cross or a yin-yang image) of size (50×50). We then generated a random tensor $\tilde{\mathcal{X}}$ of size $(1500 \times 50 \times 50)$ of which each element was sampled from a normal distribution. Finally, we constructed the corresponding response array $\mathbf{y}$ of size 1500 as: $\forall i \in \{1, \cdots, 1500\}, \mathbf{y_i} = \langle \tilde{\mathcal{X}}_i, \tilde{\mathcal{W}} \rangle$.

We use this data to train a rank-10 Kruskal Tucker Regression and a rank $(10, 10, 10)$ Tucker Ridge Regression. We also train a classical Ridge Regression model on the vectorised training samples (we use the scikit-learn implementation).

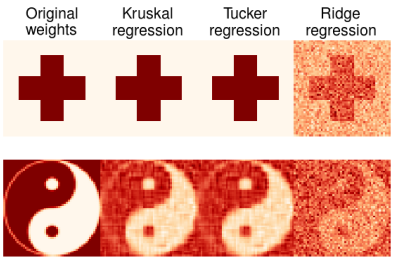

Figure 1: Example of Tensor Ridge regression compared to classical Ridge regression.

In Fig. 3.1, we present in the first column the original weight. The second and third column present the weight learnt from our Tucker and Kruskal regression models while the last column presents the weights

learned by a classical Ridge Regression. As can be observed, tensor regression gives more interpretable result due to its ability to take into account the structure in the data that is lost when vectorising it, and thanks to its low rank weight coefficient.

3.2 Tensor decomposition

We generated third order random tensors of size $(K \times K \times K)$ for K varying from 10 to 400 with a step of 10, for a total of elements varying from 1,000 to 64,000,000. We then compared runtime and performance with scikit-tensor (*sktensor*) and the Matlab Tensor Toolbox (*tensor toolbox*) for CANDECOMP-PARAFAC and Tucker decomposition of these tensors.

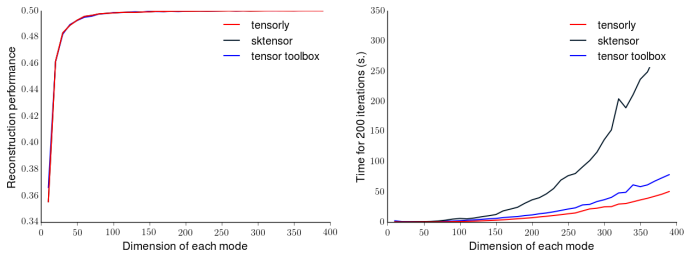

Figure 2: CANDECOMP-PARAFAC decompostion of a tensor of varying size.

We first apply a rank 10 CANDECOMP-PARAFAC decomposition via Alternating Least Squares (ALS). In Fig. 2 we show the evolution of the performance and runtime as a function of the number of elements in each mode of the tensor. Each method was run for exactly 100 iterations with an SVD based initialisation.

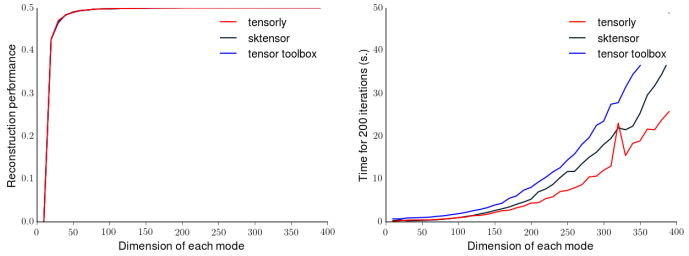

Figure 3: TUCKER decompostion of a tensor of varying size.

Similarly, in Fig. 3, we show the evolution of the performance and the execution time for a rank $(10, 10, 10)$ Tucker decomposition via Higher Order Orthogonal Iteration (HOI), when running each method for exactly 100 iterations with an SVD based initialisation.

As can be observed, all methods yield similar performance, whilst our library offers competitive speed.

4 Conclusion and future work

TensorLy makes tensor learning accessible and easy by offering state-of-the-art tensor methods and operations through simple consistent interfaces under a permissive license. Interestingly, experimental evaluation indicates that tensor decomposition implemented in *TensorLy* are executed faster than their corresponding Matlab implementation. Thus, the library allows for efficient comparison of existing methods and can be easily extended to new ones, with systematic unit-tests and documentation.

Going forward we will further extend the available decompositions with other state-of-the-art methods such as PARAFAC2, DEDICOM, etc and also include robust tensor decomposition [4, 9]. It is worth noting that proximal operators for the ℓ_1 norm and nuclear norm are already available in *TensorLy*.

References

[1] E. Acar and B. Yener. "Unsupervised Multiway Data Analysis: A Literature Survey". In: *IEEE Transactions on Knowledge and Data Engineering* 21.1 (2009), pp. 6–20.

[2] Anima Anandkumar, Rong Ge, and Majid Janzamin. "Analyzing Tensor Power Method Dynamics: Applications to Learning Overcomplete Latent Variable Models". In: *CoRR* abs/1411.1488 (2014).

[3] Animashree Anandkumar et al. "Tensor Decompositions for Learning Latent Variable Models". In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 2773–2832.

[4] Animashree Anandkumar et al. "Tensor vs Matrix Methods: Robust Tensor Decomposition under Block Sparse Perturbations". In: *CoRR* abs/1510.04747 (2015).

[5] Brett W. Bader, Tamara G. Kolda, et al. *MATLAB Tensor Toolbox Version 2.6*. Available online. 2015.

[6] A. Cichocki et al. "Tensor Decompositions for Signal Processing Applications: From two-way to multiway component analysis". In: *IEEE Signal Processing Magazine* 32.2 (2015), pp. 145–163.

[7] Andrzej Cichocki et al. *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-Way Data Analysis and Blind Source Separation*. John Wiley & Sons, Ltd, 2009.

[8] Nadav Cohen, Or Sharir, and Amnon Shashua. "On the Expressive Power of Deep Learning: A Tensor Analysis". In: *CoRR* abs/1509.05009 (2015).

[9] Donald Goldfarb and Zhiwei (Tony) Qin. "Robust Low-Rank Tensor Recovery: Models and Algorithms". In: *SIAM Journal on Matrix Analysis and Applications* 35.1 (2014), pp. 225–253.

[10] Lars Grasedyck, Daniel Kressner, and Christine Tobler. "A literature survey of low-rank tensor approximation techniques". In: *GAMM-Mitteilungen* 36.1 (2013), pp. 53–78.

[11] W. Guo, I. Kotsia, and I. Patras. "Tensor Learning for Regression". In: *IEEE Transactions on Image Processing* 21.2 (2012), pp. 816–827.

[12] J. D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science Engineering* 9.3 (2007), pp. 90–95.

[13] Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. "Beating the Perils of Non-Convexity: Guaranteed Training of Neural Networks using Tensor Methods". In: *CoRR* (2015).

[14] Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. "Score Function Features for Discriminative Learning: Matrix and Tensor Framework". In: *CoRR* abs/1412.2863 (2014).

[15] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed 2016-10-21]. 2001–. URL: `http://www.scipy.org/`.

[16] Yong-Deok Kim and Seungjin Choi. "Nonnegative Tucker Decomposition". In: *2007 IEEE Conference on Computer Vision and Pattern Recognition* (2007), pp. 1–8.

[17] Tamara G. Kolda and Brett W. Bader. "Tensor Decompositions and Applications". In: *SIAM REVIEW* 51.3 (2009), pp. 455–500.

[18] Xiaoshan Li, Hua Zhou, and Lexin Li. "Tucker tensor regression and neuroimaging analysis". In: *arXiv preprint arXiv:1304.5637* (2013).

[19] Haiping Lu, Konstantinos N. Plataniotis, and Anastasios N. Venetsanopoulos. "A survey of multilinear subspace learning for tensor data". In: *Pattern Recognition* 44.7 (2011), pp. 1540 –1551.

[20] M. Mørup. "Applications of tensor (multiway array) factorizations and decompositions in data mining". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1.1 (2011), pp. 24–40.

[21] Vervliet N. et al. *Tensorlab 3.0*. Available online. 2016.

[22] Dimitri Nion and Nicholas D. Sidiropoulos. "Tensor algebra and multidimensional harmonic retrieval in signal processing for MIMO radar". In: *IEEE Transactions on Signal Processing* 58.11 (Nov. 2010), pp. 5693–5705.

[23] Dimitri Nion et al. "Batch and Adaptive PARAFAC-based Blind Separation of Convolutive Speech Mixtures". In: *Trans. Audio, Speech and Lang. Proc.* 18.6 (2010), pp. 1193–1207.

[24] Yannis Panagakis, Constantine Kotropoulos, and Gonzalo R Arce. "Non-negative multilinear principal component analysis of auditory temporal modulations for music genre classification". In: *IEEE Transactions on Audio, Speech, and Language Processing* 18.3 (2010), pp. 576–588.

[25] Evangelos E. Papalexakis, Christos Faloutsos, and Nicholas D. Sidiropoulos. "ParCube: Sparse Parallelizable Tensor Decompositions". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part I*. Springer Berlin Heidelberg, 2012, pp. 521–536.

[26] Evangelos E. Papalexakis, Christos Faloutsos, and Nicholas D. Sidiropoulos. "Tensors for Data Mining and Data Fusion: Models, Applications, and Scalable Algorithms". In: *ACM Trans. Intell. Syst. Technol.* 8.2 (Oct. 2016), 16:1–16:44.

[27] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[28] Amnon Shashua and Tamir Hazan. "Non-negative tensor factorization with applications to statistics and computer vision". In: *In Proceedings of the International Conference on Machine Learning (ICML)*. ICML, 2005, pp. 792–799.

[29] Nicholas D Sidiropoulos et al. "tensor decomposition for signal processing and machine learning". In: *arXiv preprint arXiv:1607.01668* (2016).

[30] M. A. O. Vasilescu and Demetri Terzopoulos. "Multilinear Analysis of Image Ensembles: TensorFaces". In: *Proceedings of the 7th European Conference on Computer Vision-Part I*. ECCV '02. London, UK, UK: Springer-Verlag, 2002, pp. 447–460.

[31] S. van der Walt, S. C. Colbert, and G. Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science Engineering* 13.2 (2011), pp. 22–30.

[32] Hua Zhou, Lexin Li, and Hongtu Zhu. "Tensor regression with applications in neuroimaging data analysis". In: *Journal of the American Statistical Association* 108.502 (2013), pp. 540–552.