Journal of Machine Learning Research 23 (2022) 1-6 Submitted 10/21; Revised 5/22; Published 6/22

tntorch: Tensor Network Learning with PyTorch

Mikhail Usvyatsov[1]* MIKHAIL.USVYATSOV@GEOD.BAUG.ETHZ.CH
Rafael Ballester-Ripoll[2]* RAFAEL.BALLESTER@IE.EDU
Konrad Schindler[1] SCHINDLER@ETHZ.CH

[1]*ETH Zurich, Switzerland* [*]*Joint first authors*
[2]*IE University, Madrid, Spain*

Editor: Antti Honkela

Abstract

We present tntorch, a tensor learning framework that supports multiple decompositions (including Candecomp/Parafac, Tucker, and Tensor Train) under a unified interface. With our library, the user can learn and handle low-rank tensors with automatic differentiation, seamless GPU support, and the convenience of PyTorch's API. Besides decomposition algorithms, tntorch implements differentiable tensor algebra, rank truncation, cross-approximation, batch processing, comprehensive tensor arithmetics, and more.

Keywords: tensor decompositions, pytorch, low-rank methods, multilinear algebra

1. Introduction

In many machine learning and data analysis tasks one is faced with multi-dimensional data arrays. Tensors are a powerful tool to represent and handle such data, but often constitute a bottleneck in terms of storage and computation. *Tensor decompositions* expand a tensor into a set of separable terms. If the tensor has low rank (i.e., there are much fewer degrees of freedom than tensor elements), then such a decomposition can dramatically reduce the representation size (Kolda and Bader, 2009; Cichocki et al., 2016; Khrulkov et al., 2019).

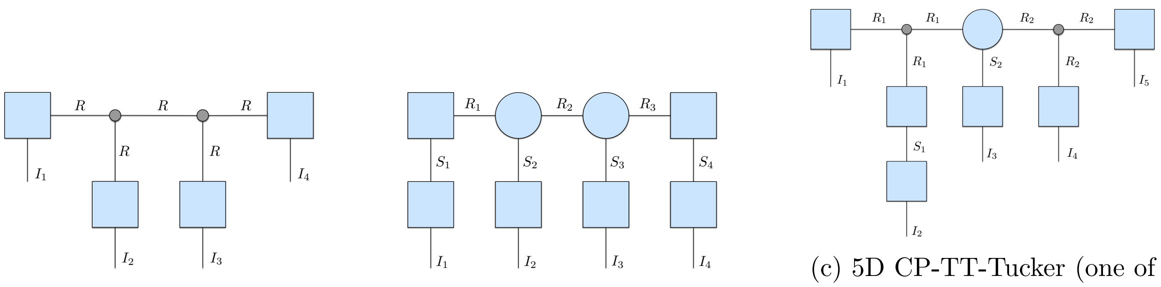
Tensor decompositions play an increasingly important role in machine learning: they are used to factorize neural network weights (Novikov et al., 2015; Gusak et al., 2019; Wang et al., 2020; Idelbayev and Carreira-Perpinán, 2020; Kossaifi et al., 2020), to accelerate learning and inference (Ma et al., 2019; Hrinchuk et al., 2019; Usvyatsov et al., 2021), to impose priors that improve accuracy (Kuznetsov et al., 2019; He et al., 2022) and robustness (Kolbeinsson et al., 2021), etc. For more applications of tensors in machine learning see (Panagakis et al., 2021). Importantly, the distinction between the different classical decomposition schemes has become blurred, as newer types of low-rank constraints and tensor indexing schemes have gained popularity (Novikov et al., 2015; Khrulkov et al., 2019; Cichocki et al., 2016; Usvyatsov et al., 2021). There exist multiple frameworks (Novikov et al., 2020; Kossaifi et al., 2019) with emphasis on different aspects. We believe that more than ever, tensor software must be flexible regarding the underlying tensor formats, while exposing a natural and accessible interface to the user.

To this end, we introduce tntorch (`github.com/rballester/tntorch`), an open-source Python package that abstracts the choice of format, while providing a wide range of tools for tensor learning, manipulation, and analysis. Compared to similar recent libraries

T3F (Novikov et al., 2020), TensorLy (Kossaifi et al., 2019), TedNet (Pan et al., 2021), TensorD (Hao et al., 2018), TensorNetwork (Roberts et al., 2019), and tt-pytorch (Khrulkov et al., 2019), tntorch emphasizes an easy-to-use, decomposition-independent interface inherited from PyTorch[1]. The package's API is fully documented and features a score of tutorial Jupyter notebooks (`tntorch.readthedocs.io`) across a variety of use cases.

2. Supported Tensor Formats

tntorch supports several decomposition models that are important in the context of machine learning, including Candedomp/Parafac (CP, Harshman, 1970), the Tucker decomposition[1] (de Lathauwer et al., 2000), and the tensor train (TT, Oseledets, 2011). These formats are encapsulated into a single class `Tensor` and share a common interface for all supported operations, which in turn replicate the API of PyTorch as closely as possible. Internally, the decomposition is stored as a low-rank *tensor network*: a graph whose nodes are low-dimensional tensors and whose edges are tensor dimensions that may be contracted together (Cichocki et al., 2016) by performing the corresponding operation (e.g., a matrix-vector product). Tensors in tntorch can mix more than one format, e.g., one may attach Tucker factors to TT cores, interleave CP and TT cores, or even blend all three formats in various ways. See Fig. 1 for illustrative examples.



R R R R R I_1 I_2 I_3 I_4
R_1 R_2 R_3 S_1 S_2 S_3 S_4 I_1 I_2 I_3 I_4
R_1 R_1 R_2 R_2 R_1 S_2 R_2 S_1 I_1 I_2 I_3 I_4 I_5

(a) 4D CP

(b) 4D TT-Tucker

(c) 5D CP-TT-Tucker (one of many possible combinations)

Figure 1: Examples of tensor networks that can be assembled in tntorch. A square represents a matrix, as used for CP factors, Tucker factors, and for the first and last cores of a TT; a circle is a 3D tensor and is used for internal TT cores; a dot is a 3D copy tensor (Biamonte, 2019), i.e., a super-diagonal tensor with ones along the diagonal.

In tntorch, every decomposition consists of a sequence of cores and, optionally, one or more factors. All nodes of the resulting tensor network may also be accessed directly if the user desires low-level control. By virtue of this abstraction, other tensor formats such as Indscal, Candelinc, Dedicom, and Paratuck2 (Kolda and Bader, 2009) can be represented as `Tensor` objects, too. See documentation for details.

1. tntorch can be installed conveniently via the PyPI package manager (command: `pip install tntorch`)
2. The Tucker tensor is represented as a TT with arbitrary ranks, which has equal expressive power. Furthermore, Tucker factors are implemented as standard unconstrained matrices, in contrast to other implementations that often implement them as unitary/orthogonal matrices.

3. Features and Operations

Learning Tensors. Given a data tensor, `tntorch` can fit a decomposition in multiple ways. E.g., gradient descent is available out-of-the-box, thanks to PyTorch's automatic differentiation and the numerous optimizers it provides. It is very general, as it can be used to learn incomplete tensors, tensors with constraints, or to add various loss terms. For the case of learning TT tensors, a cross-approximation routine is provided that performs adaptive sampling (Oseledets and Tyrtyshnikov, 2010). Other methods include *TT-SVD* (Oseledets, 2011), *higher-order SVD* and *alternating least squares* (Kolda and Bader, 2009) for the TT, Tucker and CP formats, respectively.

Tensor Arithmetics. The library supports tensor$\times$matrix and tensor$\times$vector products, element-wise operations, dot products, convolution, concatenation, mode reordering, padding, orthogonalization, rank truncation, and more. These are useful for common use cases in machine learning: tensor completion; learning low-rank layers or compressing/approximating existing layers; applying multi-linear operators; dimensionality reduction; etc. Advanced element-wise functions such as `/`, `exp`, and `tanh` can be approximately computed via cross-approximation (currently available for TT tensors). Besides these features, `tntorch` also implements further, miscellaneous tools, such as: sensitivity analysis; Boolean algebra operations; statistical moments; a smart reduction operator; sampling from TT-compressed distributions; multi-linear polynomial expansions; etc.

Slicing and Indexing. Any `Tensor` may be accessed in several ways:

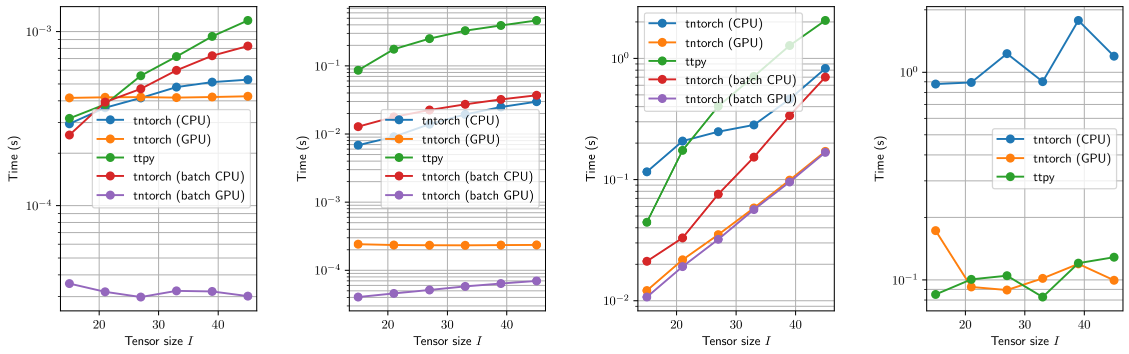
- *Basic indexing* via slices in the format `start:stop:step` and variants, as in `t[::-1]`.
- *Fancy indexing* via handcrafted, possibly irregular slices, e.g. by passing a list of indices as in `t[:, (0, 3, 4)]`.
- Indexing by NumPy arrays. For example, accessing an N-dimensional tensor by an $M \times N$ NumPy matrix returns a 1D tensor with M elements.
- Insertion of dummy dimensions, as in `t[None, :]`, or ellipsis, as in `t[..., 3]`.
- Broadcasting is supported, i.e., operations will be repeated along one or more dimensions when their sizes do not match. For example, if `t` is a 1D tensor, then `t[:, None] + t[None, :]` is a square matrix.

Such read operations match the behavior of PyTorch tensors almost exactly[3]. They are performed in the compressed domain and return a compressed `Tensor`. Moreover, tensors can be edited using the expected syntax, as in `a[:, 0, :] = 2 * b[0, :, :]` or `t[(0, 2), ..., -1, ::3] += 1`. All read/write operations preserve differentiability, a behavior at present unique to `tntorch`, to the best of our knowledge.

Batched Tensors. Batch processing is an important capability to increase the efficiency of tensor-based learning. Besides being differentiable and GPU-ready, decomposition and arithmetic operations in `tntorch` work on batches of tensors in any format; internally, a batch is represented as an extra dimension in every node of the tensor network. As an example, we can decompose multiple tensors into TT cores at once using the TT-SVD algorithm (Oseledets, 2011), with linear algebra operations applied in batches. See Sec. 4 for a benchmark.

Tensor Train Matrices. The TT matrix decomposition has recently become a particularly relevant format, because it is, among others, well-suited to compress neural weight

1. Interleaving fancy and basic indices is not supported, as it is in general computationally expensive.



(a) Element-wise sum (b) Element-wise prod. (c) TT-SVD (d) Cross-approx.

Figure 2: Runtimes of four operations with varying tensor sizes (average time per processed object, over 10 runs). The batch size is set to $B = 32$.

layers (Novikov et al., 2015). Unlike standard tensors, a TT matrix has a different indexing scheme, with two sets of row and column indices. As a consequence, TT matrices come with a distinct set of computation rules for standard operations such as matrix$\times$vector multiplication. Therefore, they have their own class `TTMatrix`, with a separate API.

4. Performance

We benchmark `tntorch`'s running times across four representative operations: TT decomposition using the TT-SVD algorithm, cross-approximation, and two arithmetic operations that can be achieved by direct manipulation of TT cores (Oseledets, 2011). We test four modalities: CPU vs. GPU, and in both cases *for loop* vs. vectorized batch processing. As a baseline, we also compare with the Python library ttpy (Oseledets, 2015), which is written in NumPy and FORTRAN and also implements these four operations. All experiments use randomly initialized tensors of TT-rank $R = 20$, physical dimension sizes $I = 15, \dots, 45$, and number of dimensions $N = 8$ (except for the TT-SVD experiment, where $N = 4$). We used PyTorch 1.13.0a0+git87148f2 (compiled from source) and NumPy 1.22.4 on an Intel(R) Core(TM) i7-7700K CPU with 64Gb RAM and an NVIDIA GeForce RTX 3090 GPU.

Results are reported in Fig. 2. Note that the GPU is more performant on both batch and non-batch modes. Also, `tntorch` scales better (or similarly, for cross-approximation) than the baseline w.r.t. the tensor size, and is thus a good fit for data-intensive ML applications.

5. Conclusions

We have introduced `tntorch`, a PyTorch-powered library that unifies multiple efficient tensor formats under the same interface, together with a rich suite of learning and analysis routines. The library comes with many standard features of modern machine learning frameworks, including auto-differentiation, GPU and batch processing, and advanced indexing. `tntorch` imitates the look and feel of standard PyTorch tensors, while making the power of low-rank tensor decompositions accessible for machine learning.

References

Jacob Biamonte. Lectures on quantum tensor networks. *arXiv preprint arXiv:1912.10049*, 2019.

Andrzej Cichocki, Namgil Lee, Ivan V. Oseledets, Anh-Huy Phan, Qibin Zhao, and Danilo P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 - Low-rank tensor decompositions. *Foundations and Trends in Machine Learning*, 9 (4-5):249–429, 2016.

Lieven de Lathauwer, Bart de Moor, and Joos Vandewalle. On the best rank-1 and rank-$(R_1, R_2, ..., R_N)$ approximation of higher-order tensors. *SIAM Journal of Matrix Analysis and Applications*, 21(4):1324–1342, 2000.

Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Ivan Oseledets, and Andrzej Cichocki. MUSCO: Multi-stage compression of neural networks. *arXiv preprint 1903.09973*, 2019.

Liyang Hao, Siqi Liang, Jinmian Ye, and Zenglin Xu. TensorD: A tensor decomposition library in TensorFlow. *Neurocomputing*, 318:196–200, 2018.

Richard A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

Wei He, Yong Chen, Naoto Yokoya, Chao Li, and Qibin Zhao. Hyperspectral super-resolution via coupled tensor ring factorization. *Pattern Recognition*, 122:108280, 2022.

Oleksii Hrinchuk, Valentin Khrulkov, Leyla Mirvakhabova, Elena Orlova, and Ivan Oseledets. Tensorized embedding layers for efficient model compression. *arXiv preprint 1901.10787*, 2019.

Yerlan Idelbayev and Miguel A Carreira-Perpinán. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8049–8059, 2020.

Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, Elena Orlova, and Ivan Oseledets. Tensorized embedding layers for efficient model compression. *arXiv preprint 1901.10787*, 2019.

Arinbjörn Kolbeinsson, Jean Kossaifi, Yannis Panagakis, Adrian Bulat, Animashree Anandkumar, Ioanna Tzoulaki, and Paul M Matthews. Tensor dropout for robust learning. *IEEE Journal of Selected Topics in Signal Processing*, 15(3):630–640, 2021.

Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. TensorLy: Tensor learning in Python. *Journal of Machine Learning Research*, 20(1):925–930, 2019.

Jean Kossaifi, Antoine Toisoul, Adrian Bulat, Yannis Panagakis, Timothy M Hospedales, and Maja Pantic. Factorized higher-order CNNs with an application to spatio-temporal emotion estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6060–6069, 2020.

Maksim Kuznetsov, Daniil Polykovskiy, Dmitry Vetrov, and Alexander Zhebrak. A prior of a googol Gaussians: a tensor ring induced prior for generative models. In *Advances in Neural Information Processing Systems*, pages 4104–4114, 2019.

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. A tensorized transformer for language modeling. In *Advances in Neural Information Processing Systems*, pages 2232–2242, 2019.

A. Novikov, P. Izmailov, V. Khrulkov, M. Figurnov, and I. Oseledets. Tensor train decomposition on TensorFlow (T3F). *Journal of Machine Learning Research*, 21:1–7, 2020.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

Ivan V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33 (5):2295–2317, 2011.

Ivan V. Oseledets. ttpy: Python implementation of the tensor train (TT)-Toolbox. GitHub repository, `https://github.com/oseledets/ttpy`, 2015.

Ivan V. Oseledets and Evgeny Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra Applications*, 432(1):70–88, 2010.

Yu Pan, Maolin Wang, and Zenglin Xu. TedNet: a PyTorch toolkit for tensor decomposition networks. *arXiv e-prints*, 2021.

Yannis Panagakis, Jean Kossaifi, Grigorios G Chrysos, James Oldfield, Mihalis A Nicolaou, Anima Anandkumar, and Stefanos Zafeiriou. Tensor methods in computer vision and deep learning. *Proceedings of the IEEE*, 109(5):863–890, 2021.

Chase Roberts, Ashley Milsted, Martin Ganahl, Adam Zalcman, Bruce Fontaine, Yijian Zou, Jack Hidary, Guifre Vidal, and Stefan Leichenauer. TensorNetwork: A library for physics and machine learning, 2019.

Mikhail Usvyatsov, Anastasia Makarova, Rafael Ballester-Ripoll, Maxim Rakhuba, Andreas Krause, and Konrad Schindler. Cherry-picking gradients: Learning low-rank embeddings of visual data via differentiable cross-approximation. In *International Conference on Computer Vision*, 2021.

Dingheng Wang, Bijiao Wu, Guangshe Zhao, Hengnu Chen, Lei Deng, Tianyi Yan, and Guoqi Li. Kronecker cp decomposition with fast multiplication for compressing rnns. *arXiv preprint 2008.09342*, 2020.