Tensor Train Decomposition on TensorFlow (T3F)

Alexander Novikov[1,2]   SASHA.V.NOVIKOV@GMAIL.COM
Pavel Izmailov[3]   PI49@CORNELL.EDU
Valentin Khrulkov[4]   VALENTIN.KHRULKOV@SKOLKOVOTECH.RU
Michael Figurnov[1]   MICHAEL@FIGURNOV.RU
Ivan Oseledets[2,4]   I.OSELEDETS@SKOLTECH.RU

[1] *National Research University Higher School of Economics, Moscow, Russia*

[2] *Institute of Numerical Mathematics RAS, Moscow, Russia*

[3] *Cornell University, Ithaca NY, USA*

[4] *Skolkovo Institute of Science and Technology, Moscow, Russia*

Abstract

Tensor Train decomposition is used across many branches of machine learning. We present T3F—a library for Tensor Train decomposition based on TensorFlow. T3F supports GPU execution, batch processing, automatic differentiation, and versatile functionality for the Riemannian optimization framework, which takes into account the underlying manifold structure to construct efficient optimization methods. The library makes it easier to implement machine learning papers that rely on the Tensor Train decomposition. T3F includes documentation, examples and 94% test coverage.

Keywords: tensor decomposition, tensor train, software, gpu, tensorflow

1. Introduction

Methods based on tensor decompositions have become ubiquitous in the machine learning community. They are used for analyzing theoretical properties of deep networks (Cohen et al., 2016; Cohen and Shashua, 2016; Khrulkov et al., 2017; Janzamin et al., 2015), network compression (Lebedev et al., 2015; Novikov et al., 2015; Yu et al., 2017), training probabilistic models (Anandkumar et al., 2012; Jernite et al., 2013; Song et al., 2013), parametrizing recommender systems (Frolov and Oseledets, 2017), and many more. In this work, we present a library for a particular tensor decomposition, namely, the Tensor Train decomposition (Oseledets, 2011).

The Tensor Train (TT) format (Oseledets, 2011) (which is also called the Matrix Product State (MPS) format in the physics community (Fannes et al., 1992)) is a generalization of matrix low-rank format to higher-dimensional tensors. A tensor $A \in \mathbb{R}^{n^d}$ is said to be represented in the TT-format if each of its elements can be represented as a product of factors

$$A_{i_1\ldots i_d} = \sum_{\alpha_1=1}^{r} \cdots \sum_{\alpha_{d-1}=1}^{r} G^1_{i_1\alpha_1} G^2_{\alpha_1 i_2\alpha_2} G^3_{\alpha_2 i_3\alpha_3} \ldots G^d_{\alpha_{d-1}i_d},$$

where $G^1 \in \mathbb{R}^{n \times r}$, $G^2, \ldots, G^{d-1} \in \mathbb{R}^{r \times n \times r}$, and $G^d \in \mathbb{R}^{r \times n}$ are called *TT-cores*. The number of values r the auxiliary indices α_k can take is called *TT-rank*. It controls the trade-off between memory/computational efficiency and the representational power of the format. Representing tensor A via explicit enumeration of its elements requires n^d scalars, while representing it in the TT-format only requires $rn(r(d-2)+2)$.

The Tensor Train decomposition is supported by many libraries (see Section 2), but some of the recent papers that use it for machine learning purposes had to rewrite the core functionality from scratch. We believe that this is due to the lack of a library with simultaneous support of the following features important for machine learning research: GPU execution, automatic differentiation, parallel processing of a batch of tensors, and advanced support for Riemannian geometry operations—a technique for speeding up the optimization when the parameters are constrained to have a compact Tensor Train (TT) representation (or other constraint sets that form a smooth manifold).

In the presented library, we aim to make the results of the recent machine learning papers utilizing the TT-format easy to reproduce and provide a flexible framework for developing new ideas. The library is released[1] under MIT license and is distributed as a PyPI package[2] to simplify the installation process. The documentation is also available online[3]. The library includes Jupyter notebook examples, e.g. for performing tensor completion by assuming that the result has a low TT-rank and minimizing the residual between the tensor and the known elements by gradient descent or by Riemannian optimization. The library supports both graph and eager TensorFlow execution, making it easy to prototype, debug and deploy the resulting models. The library has 94% test coverage.

2. Related Work

Many libraries implement operations for working with the TT-format. The most similar ones to the presented library (T3F) are tntorch (Ballester-Ripoll, 2018) and TensorLy (Kossaifi et al., 2019). The main differences between T3F and these libraries are that tntorch uses PyTorch as a backend, T3F uses TensorFlow, and TensorLy supports PyTorch, TensorFlow and MxNet as backends. Another difference is that neither tntorch nor TensorLy support Riemannian optimization operations.

Other libraries include: ttpy (Oseledets, 2012), mpnum (Suess and Holzäpfel, 2017), scikit_tt (Gel et al., 2018), mpys (Alvarorga, 2017), OSMPS (Jaschke et al., 2018), evoMPS (Milsted, 2011), TT-Toolbox (Oseledets et al., 2011), ITensor (Matthew Fishman and White, 2013), libtt (Matveev, 2013) and MPS (Ripoll, 2013). Unlike these libraries, T3F supports GPU execution, automatic differentiation (including Riemannian autodiff) and batch processing. However, some of these libraries support advanced algorithms such as DMRG (Khoromskij and Oseledets, 2010) or AMen (Dolgov and Savostyanov, 2014), which T3F does not.

---

[1]https://github.com/Bihaqo/t3f
[2]https://pypi.python.org/pypi/t3f
[3]https://t3f.readthedocs.io

3. Implementation Details

The library provides two base classes: `TensorTrain` and `TensorTrainBatch` that support storing one tensor in the Tensor Train format and a batch of such tensors respectively, i.e. a list of tensors of the same shape that are supposed to be processed together. These two classes support most of the API of `tf.Tensor` class (e.g. `.op`, `.name`, and `.get_shape()`). Under the hood, these classes are containers for the TT-cores represented as `tf.Tensor` objects, plus lightweight meta-information. This design allows one to work with the TT-cores directly if necessary. The rest of the library is a collection of functions, each taking as an input one or two TT-objects and outputting a TT-object or a numerical value depending on the semantics of the function. For example, the function `t3f.multiply(left, right)` implements elementwise multiplication of two TT-tensors (or batches of TT-tensors) but also supports multiplication of a TT-tensor by a number. As an output, this function returns a `TensorTrain` or a `TensorTrainBatch` object.

A typical machine learning use case would be to initialize a random TT-matrix and use it to define a neural network layer (using `t3f.matmul(tt_matrix, x)` to compute the matrix-by-vector product instead of the usual `tf.matmul(matrix, x)`). Then, one can use the optimization functionality of TensorFlow to train the model with respect to its parameters including the TT-cores of the TT-matrix (Novikov et al., 2015; Stoudenmire and Schwab, 2016; Lebedev et al., 2015; Yang et al., 2017). See this and other example in the documentation[4].

A typical non-machine-learning use case would be finding the solution of a PDE discretized on a grid in the TT-format (Grasedyck et al., 2013; Khoromskij, 2018). One would initialize a tensor in the TT-format and then iteratively perform the updates as a sequence of operations on the tensor. At the end of each iteration, one would typically use the rounding operation `t3f.round(iterate, max_tt_rank=r)`, which finds the best approximation to a TT-tensor among TT-tensors of lower TT-rank, in order to restrict the TT-rank growth. T3F is a good fit for these applications because of the Riemannian optimization and GPU support and it was already used by Rakhuba et al. (2019).

3.1. Batch Processing

Most operations accept broadcasting and getting a batch of TT-objects as an input. For example, `C = t3f.matmul(A, B)` for a batch of TT-matrices `A` and a TT-matrix `B` will return a batch of TT-matrices `C` where $C_i = A_i B$ and the result is computed in parallel across the batch dimension. This functionality is important for efficient mini-batch processing of large scale data sets.

3.2. Riemannian Geometry Operations

One of the advantages of the Tensor Train format is that the set of tensors of fixed TT-rank forms a Riemannian manifold, which allows the use of Riemannian geometry ideas to speed up tensor calculus while preserving theoretical guaranties (see Steinlechner (2016) for more details). The T3F library has rich support for Riemannian operations, the most basic

---

[4]`https://t3f.readthedocs.io/en/latest/tutorials/tensor_nets.html`

being projecting a TT-object A (or a batch of them) onto the tangent space of another TT-object B. We denote this projection operation by $P_B A$.

Other supported operations are special cases of combining this basic projection operation with non-Riemannian operations but are heavily optimized by exploiting the structure of objects that are projected onto the same tangent space. Such operations include projecting a weighted sum of a batch of TT-objects on a tangent space (used for efficiently computing the Riemannian gradient on a mini-batch of objects). Mathematically, this function implements projecting a weighted sum of tensors $S = P_B \left(\sum_i c_i A_i \right)$. The same operation can be implemented by a summation followed by a projection operation in asymptotic complexity $\mathcal{O}(bdr_A r_B n(r_A b + r_B))$, where b is the batch-size and r_A and r_B are the TT-ranks of the tensors A and B. But the tailored `project_sum` operation requires only $\mathcal{O}(bdr_A r_B n(r_A + r_B))$ for the same operation. The idea behind `project_sum` is to use the linearity of the projection $S = \sum_i c_i P_B A_i$, and the fact that elements of the same tangent space can be added efficiently. Other tailored operations include computing the Gram matrix of a batch of tensors from the same tangent space and projecting the matrix-by-vector product onto a tangent space $P_b(Ac)$.

Additionally, T3F is the only library that supports automatic Riemannian differentiation for TT-format, i.e. given a function $f(X)$ the library can automatically (and with optimal asymptotic complexity) find the projection of the gradient of the function on the tangent space of the tensor X: $P_X \frac{\partial f}{\partial X}$. The product $P_{T_X} \left(\nabla^2 f(X) \right) P_{T_X} Z$ of the linearized Riemannian Hessian by a given tensor Z is also supported.

4. Benchmarking

We benchmark the basic functionality of T3F on CPU and GPU and compare its performance against an alternative library TTPY. To reproduce the benchmark on your hardware, see `docs/benchmark` folder in the T3F library.

For benchmarking, we generated a batch of 100 random TT-matrices of sizes $10^{10} \times 10^{10}$ (so $d = 10$ and the TT-representation consists of 10 TT-cores) of TT-rank 10 and a batch of 100 random TT-vectors of size $10^{10} \times 1$. We benchmarked the matrix-by-vector multiplication ('matvec'), matrix-by-matrix multiplication ('matmul'), computing the Frobenius norm ('norm'), and computing the Gram matrix of 1 or 100 TT-vectors. The results are reported in Tbl. 1. We report that T3F is faster than TTPY for most operation and that batch processing and GPU acceleration yield significant speedups for some operations (Tbl. 1). Note that TTPY lacks GPU and batch processing support.

Acknowledgments

Op	TTPY 1 object CPU	T3F 1 object CPU	T3F 1 object GPU	T3F 100 objects CPU	T3F 100 objects GPU
matvec	11.142	1.190	0.744	1.885	0.140
matmul	86.191	9.849	0.950	17.483	1.461
norm	3.790	2.136	1.019	0.253	0.044
gram	0.145	0.606	0.973	0.021	0.001
project	116.868	3.001	13.239	1.645	0.226

Table 1: Time in ms for different operations implemented in TTPY library vs T3F on CPU and GPU. The timing for a batch of 100 objects is reported per single object. The comparison is made on an NVIDIA DGX-1 station with Tesla V100 GPUs (using only 1 GPU at a time) in double precision.

References

Alvarorga. *mpys: Library for MPS computations in Python*, 2017. URL `https://github.com/alvarorga/mpys`.

A. Anandkumar, D. P. Foster, D. Hsu, S. M. Kakade, and L. Yi-kai. A spectral algorithm for latent dirichlet allocation. In *Advances in Neural Information Processing Systems 25*, pages 917–925. 2012.

Rafael Ballester-Ripoll. *tntorch: Tensor Network Learning with PyTorch*, 2018. URL `https://github.com/rballester/tntorch`.

N. Cohen and A. Shashua. Convolutional rectifier networks as generalized tensor decompositions. In *International Conference on Machine Learning*, pages 955–963, 2016.

N. Cohen, O. Sharir, and A. Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016.

Sergey V Dolgov and Dmitry V Savostyanov. Alternating minimal energy methods for linear systems in higher dimensions. *SIAM Journal on Scientific Computing*, 36(5):A2248–A2271, 2014.

M. Fannes, B. Nachtergaele, and R.F. Werner. Finitely correlated states on quantum spin chains. *Comm. Math. Phys.*, 144(3):443–490, 1992. ISSN 0010-3616. doi: 10.1007/BF02099178.

E. Frolov and I. Oseledets. Tensor methods and recommender systems. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(3), 2017.

Patrick Gel, Stefan Klus, Martin Scherer, and Feliks Nske. *scikit_tt: Tensor Train Toolbox*, 2018. URL `https://github.com/PGelss/scikit_tt`.

Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.

M. Janzamin, H. Sedghi, and A. Anandkumar. Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods. In *Advances in Neural Information Processing Systems 28*. 2015.

Daniel Jaschke, Michael L Wall, and Lincoln D Carr. Open source matrix product states: Opening ways to simulate entangled many-body quantum systems in one dimension. *Computer Physics Communications*, 225:59–91, 2018.

Y. Jernite, Y. Halpern, and D. Sontag. Discovering hidden variables in noisy-or networks using quartet tests. In *Neural Information Processing Systems (NIPS)*, 2013.

B. N. Khoromskij and I. V. Oseledets. DMRG+QTT approach to computation of the ground state for the molecular Schrödinger operator. Preprint 69, MPI MIS, Leipzig, 2010.

Boris N Khoromskij. *Tensor numerical methods in scientific computing*, volume 19. Walter de Gruyter GmbH & Co KG, 2018.

V. Khrulkov, A. Novikov, and I. Oseledets. Expressive power of recurrent neural networks. *arXiv preprint arXiv:1711.00811*, 2017.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research*, 20(26):1–6, 2019.

V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

E. Miles Stoudenmire Matthew Fishman and Steven R. White. *itensor: An implementation of the time dependent variational principle for matrix product states*, 2013. URL `http://itensor.org/`.

Sergey Matveev. *libtt library*, 2013. URL `https://bitbucket.org/matseralex/tt_smoluh/src/master/libtt/`.

Ashley Milsted. *evoMPS: An implementation of the time dependent variational principle for matrix product states*, 2011. URL `https://github.com/amilsted/evoMPS`.

A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

I. V. Oseledets. Tensor-Train decomposition. *SIAM J. Scientific Computing*, 33(5):2295–2317, 2011.

Ivan Oseledets. *ttpy: Python implementation of the TT-Toolbox*, 2012. URL `https://github.com/oseledets/ttpy`.

Ivan Oseledets, Sergey V. Dolgov, Alexey Boyko, Dmitry Savostyanov, Alexander Novikov, and Thomas Mach. *TT-Toolbox: Matlab implementation of Tensor Train decomposition*, 2011. URL `https://github.com/oseledets/TT-Toolbox`.

Maxim Rakhuba, Alexander Novikov, and Ivan Oseledets. Low-rank riemannian eigensolver for high-dimensional hamiltonians. *Journal of Computational Physics*, 396:718–737, 2019.

Juan Jos Garca Ripoll. *mps: Library for the simulation of one-dimensional quantum many-body systems*, 2013. URL `https://github.com/juanjosegarciaripoll/mps`.

L. Song, M. Ishteva, A. Parikh, E. Xing, and H. Park. Hierarchical tensor decomposition of latent tree graphical models. In *International Conference on Machine Learning (ICML)*, 2013.

M. Steinlechner. Riemannian optimization for solving high-dimensional problems with low-rank tensor structure. 2016.

Edwin Stoudenmire and David J Schwab. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems 29*. 2016.

Daniel Suess and Milan Holzäpfel. mpnum: A matrix product representation library for Python. *Journal of Open Source Software*, 2(20):465, 2017. ISSN 2475-9066. doi: 10. 21105/joss.00465. URL `https://doi.org/10.21105/joss.00465`.

Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3891–3900. JMLR. org, 2017.

R. Yu, S. Zheng, A. Anandkumar, and Y. Yue. Long-term forecasting using tensor-train rnns. *arXiv preprint arXiv:1711.00073*, 2017.