The Journal of Open Source Software

torchTT: A PyTorch-based Tensor-Train Toolbox

Ion Gabriel Ion [1]

1 Terra Quantum AG

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 16 January 2026
Published: unpublished

Summary

`torchTT` is a Python library enabling Tensor-Train (TT) decomposition (Oseledets, 2011) within the PyTorch framework (Paszke et al., 2019). It allows users to create and manipulate high-dimensional tensors in compressed TT format using PyTorch-like syntax and semantics. Operations such as basic linear algebra, matrix-vector products, and Kronecker products can be performed directly on the compressed representation without ever forming the full tensor. All computations leverage PyTorch's capabilities: TT instances live on the same device as their underlying data, facilitating seamless GPU acceleration, and automatic differentiation works through TT operations. Under the hood, `torchTT` exploits low-rank structure in high-dimensional arrays, dramatically reducing memory and computation compared to dense tensors. These features make `torchTT` especially suitable for problems with millions or billions of degrees of freedom where explicit storage is impossible.

Statement of Need

High-dimensional tensors arise in many areas of science and engineering, such as the discretization of d-dimensional PDEs, state spaces for ODE systems, and many-body wavefunctions in quantum chemistry. Naively storing or computing with a full tensor scales exponentially in d (the curse of dimensionality). The TT decomposition combats this by factorizing a tensor into a sequence of smaller, low-rank 3D cores.

However, existing TT software (e.g., `ttpy`, `TensorLy` (Kossaifi et al., 2019)) often lacks tight integration with modern machine learning tools or specialized solvers. `torchTT` addresses this gap by providing a native PyTorch implementation. Compared to `ttpy`, it offers native GPU acceleration and automatic differentiation, enabling seamless integration with deep learning workflows. In contrast to `tntorch` and `TensorLy`, `torchTT` provides specialized capabilities for solving linear systems and constructing tensors from function evaluations (cross-approximation). This combination of features makes it particularly suitable for scientific machine learning, where one must frequently compute with compressed tensors or solve large linear systems directly in TT format.

Functionality and Features

The `torchTT` library is organized into key modules enabling specific functionality on TT-tensors:

- **Core TT Class**: The `torchtt.TT` class is the central component, handling both TT-tensors and TT-matrices. It implements a comprehensive suite of linear algebra operations, ranging from basic elementwise operations (addition, multiplication) to complex reductions (dot products, norms). Crucially, it supports "fast algebra": operations like matrix-vector products and elementwise multiplication can be performed using efficient DMRG/AMEn-based algorithms that adaptively determine the optimal rank, avoiding

intermediate rank explosions. The class also supports slicing, concatenation, and reshaping, allowing users to manipulate compressed tensors with the same ease as dense PyTorch tensors.

- **Linear System Solvers**: The `torchtt.solvers` module implements advanced TT solvers, notably the Alternating Minimal Energy (AMEn) method (Dolgov & Savostyanov, 2014) for solving linear systems $Ax = b$ directly in the TT format. The solver enriches TT ranks dynamically to achieve high accuracy. Other routines include elementwise inversion and division using AMEn-based iterative schemes.
- **Cross Approximation**: The `torchtt.interpolate` module implements adaptive TT-cross methods (Oseledets, 2010) to build TT representations from function evaluations. This allows constructing TT approximations of multivariate functions (e.g., $y = \log(x)$ elementwise) without forming full grids, using alternating sweeps that sample tensor entries.
- **Neural Network Layers**: `torchtt.nn` provides TT-format neural network layers, such as `LinearLayerTT`. These layers parametrize weights as TT-matrices, significantly reducing parameter counts and memory footprint compared to dense layers while allowing end-to-end training via standard PyTorch backpropagation.
- **Manifold and Optimization**: The `torchtt.manifold` module provides tools for Riemannian optimization on the manifold of tensors with fixed TT-rank, including projections onto the tangent space. Combined with PyTorch's autograd, this facilitates gradient-based optimization on TT-manifolds.

Example: Solving a 4D PDE

The following example demonstrates how to solve a 4-dimensional Poisson equation $\Delta u = f$ on $[0, 1]^4$ with zero boundary conditions using `torchTT`. The right-hand side f is constructed using cross-approximation, and the system is solved directly in the TT format using the AMEn solver.

```
import torchtt as tntt
import torch as tn

# Define the problem size
N, d = 64, 4

# Construct the 1D Laplacian operator (finite difference)
L1d = (tn.diag(tn.ones(N-1),-1) + tn.diag(tn.ones(N-1),1) - \
      2*tn.eye(N)) / (1/(N-1))**2
L1d[0,1] = L1d[-1,-2] = 0 # Boundary conditions
L1d_tt = tntt.TT(L1d, shape=[(N,N)])

# Construct the d-dimensional Laplacian
L = L1d_tt ** tntt.eye([N]*3) + tntt.eye([N]) ** L1d_tt ** tntt.eye([N]*2) + \
    tntt.eye([N]*2) ** L1d_tt ** tntt.eye([N]) + tntt.eye([N]*3) ** L1d_tt

# Create the grid for cross approximation
x = tntt.meshgrid([tn.linspace(0,1,N)]*d)

# Approximate the RHS f(x) = x1(x1-1)*...*xd(xd-1)
f = tntt.interpolate.function_interpolate(lambda x: tn.prod(x * (x - 1), dim=1), x, eps=

# Solve the linear system Ax = f using AMEn on GPU
u = tntt.solvers.amen_solve(L.to('cuda'), f.to('cuda'), eps=1e-6)
```

Research Impact and Applications

torchTT opens up new possibilities for research in domains plagued by high dimensionality:

- **High-Dimensional PDEs/ODEs**: torchTT can solve PDEs and dynamical systems in high dimensions by representing solution spaces and operators in TT form. Built-in solvers like AMEn make it practical to compute time steps of multi-dimensional PDEs, breaking the curse of dimensionality.
- **Computational Physics and Chemistry**: The library enables efficient representation of high-dimensional objects like quantum many-body wavefunctions. Operators can be applied via TT-matrix-vector products, and observables computed via TT inner products, facilitating large-scale simulations.
- **Machine Learning with Compressed Models**: TT-layers enable significant compression of model parameters in deep learning. torchTT's compatibility with autograd allows for end-to-end training of compressed architectures and efficient on-device inference.
- **Data Science and Function Approximation**: The library is useful for surrogate modeling and uncertainty quantification, where high-dimensional response surfaces can be approximated adaptively using cross-approximation routines.

AI Usage Disclosure

Artificial Intelligence tools were used to rephrase and refine the text of this paper. The core software implementation and the original draft of the content were produced by the authors.

References

Dolgov, S. V., & Savostyanov, D. V. (2014). Alternating minimal energy methods for linear systems in higher dimensions. *SIAM Journal on Scientific Computing*, *36*(5), A2248–A2271. https://doi.org/10.1137/140953289

Kossaifi, J., Panagakis, Y., Anandkumar, A., & Pantic, M. (2019). TensorLy: Tensor learning in python. *Journal of Machine Learning Research*, *20*(26), 1–6. http://jmlr.org/papers/v20/18-277.html

Oseledets, I. V. (2010). TT-cross approximation for multidimensional arrays. *Linear Algebra and Its Applications*, *432*(1), 70–88. https://doi.org/10.1016/j.laa.2009.07.024

Oseledets, I. V. (2011). Tensor-train decomposition. *SIAM Journal on Scientific Computing*, *33*(5), 2295–2317. https://doi.org/10.1137/090752286

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf