Journal of Machine Learning Research 25 (2024) 1-7 Submitted 5/23; Revised 1/24; Published 3/24

ptwt - The PyTorch Wavelet Toolbox

Moritz Wolter MORITZ.WOLTER@UNI-BONN.DE
High-Performance Computing and Analytics Lab, University of Bonn, Germany

Felix Blanke FELIX.BLANKE@SCAI.FRAUNHOFER.DE
Fraunhofer Institute for Algorithms and Scientific Computing, Sankt Augustin, Germany

Jochen Garcke GARCKE@INS.UNI-BONN.DE
Institute for Numerical Simulation, University of Bonn
and Fraunhofer Institute for Algorithms and Scientific Computing, Sankt Augustin, Germany

Charles Tapley Hoyt CTHOYT@GMAIL.COM
Northeastern University, Boston, USA

Editor: Sebastian Schelter

Abstract

The fast wavelet transform is an important workhorse in signal processing. Wavelets are local in the spatial- or temporal- and the frequency-domain. This property enables frequency domain analysis while preserving some spatiotemporal information. Until recently, wavelets rarely appeared in the machine learning literature. We provide the PyTorch Wavelet Toolbox to make wavelet methods more accessible to the deep learning community. Our PyTorch Wavelet Toolbox is well documented. A pip package is installable with `pip install ptwt`.

Keywords: PyTorch, wavelet, wavelet-packets, wavelet-analysis, wavelet-transform

1. Introduction

Nowadays, wavelets are used to extract information from many different kinds of data, with a particular focus on audio signals and images. They are similar to Fourier analysis since a signal is decomposed, but wavelets are localized in time –or space– and frequency, which means that they can capture information about a signal at different scales and resolutions. This is useful for analyzing signals that contain both high-frequency and low-frequency components, such as speech or images (Torrence and Compo, 1998). The Fast Wavelet Transform (FWT) is an algorithm to perform the wavelet transform on a digital signal in an efficient and computationally feasible manner, it has a long and proven track record as an excellent tool in engineering and science (Mallat, 2008). For further background on wavelets, we refer to the excellent textbooks by Strang and Nguyen (1996), Jensen and la Cour-Harbo (2001), and Daubechies (1992). While initially introduced for signal processing tasks, the wavelet transform has started to appear in machine learning contexts. Some notable tasks include deepfake detection (Huang et al., 2022; Gasenzer and Wolter, 2023) and neural network compression (Wolter et al., 2020). At the intersection of signal processing and neural network design Recoskie (2018) explored wavelet filter learning, while Cotter (2020) studied the application of complex wavelets in neural networks. Major popular machine learning frameworks like PyTorch (Paszke et al., 2017, 2019) and JAX (Bradbury et al., 2018) lack native Fast Wavelet Transform (FWT)-support. In the Python ecosystem, separate frameworks like PyWavelets (Lee et al., 2019) and "2D Wavelet Transforms in Pytorch" (Cotter, 2022, 2020) exist. Lee et al. (2019) focus on CPU support and provide an extensive library of precomputed wavelet filters. Cotter (2022) supports the padded separable two-dimensional wavelet transform and its complex dual-tree variant. Both focus on padded transforms. To our knowledge, we are proposing the first toolbox with boundary wavelet support. The presented code adds Graphics Processing Unit (GPU) and gradient support for single- and three-dimensional transforms and the fully separable wavelet transform. Toolbox and documentation are available online. [1]

2. Library Design

Our library builds on the PyWavelets (`pywt`) package (Lee et al., 2019). Among other features, we add boundary-wavelet as well as automatic differentiation, and Just In Time Compilation (jit) support. Our package is available for user-friendly installation via,

```
pip install ptwt
```

We reuse the `pywt.Wavelet` data type for access to an extensive collection of predefined wavelet filters. We have worked hard to make both Application Programming Interfaces (sAPIs) as compatible as possible. In many cases, migrating from `pywt` to `ptwt` or the other way around requires only a transfer of the data into a `torch.Tensor` or `numpy.ndarray` format. The code snipped below illustrates the similarities.

```python
import torch
import pywt, ptwt
# generate an input of even length.
data = torch.tensor([0., 1., 2., 3., 4., 5.])
# compare the forward fwt coefficients
print(pywt.wavedec(data.numpy(), "db2", mode="zero", level=2))
print(ptwt.wavedec(data, "db2", mode="zero", level=2))
# invert the fwt
print(ptwt.waverec(ptwt.wavedec(data, "db2", mode="zero"), "db2"))
```

In addition to padded transforms, which all libraries allow, we provide support for boundary wavelet filters (Strang and Nguyen, 1996). Instead of padding the edges, boundary filter transforms use orthogonalized analysis and synthesis matrices. Efficient orthogonalization relies on a QR decomposition, which is available natively in PyTorch.

At the time of writing, our unit tests ensure Python 3.9 and 3.11 compatibility. Older versions may run as well, and we intend to provide support for additional future versions when they become available. We may deprecate older versions when we do. We provide examples illustrating possible applications of wavelets in machine learning, like deepfake identification (Wolter et al., 2022) or wavelet optimization (Wolter and Garcke, 2021).

1. `https://pypi.org/project/ptwt/`, `https://pytorch-wavelet-toolbox.readthedocs.io/en/latest/`

		run-time [s]		
		ours	Cotter (2022)	Lee et al. (2019)
DWT-1D	CPU	0.40286 ± 0.00638	-	0.25841 ± 0.00907
	GPU	0.00887 ± 0.04413	-	-
	GPU-jit	**0.00439 ± 0.00051**	-	-
DWT-2D	CPU	0.17453 ± 0.01335	-	0.54936 ± 0.00924
	GPU	0.01447 ± 0.03995	-	-
	GPU-jit	**0.01110 ± 0.00050**	-	-
DWT-2D-sep.	CPU	0.52484 ± 0.00790	0.40189 ± 0.00727	0.92772 ± 0.00295
	GPU	0.00995 ± 0.00062	0.01474 ± 0.04667	-
	GPU-jit	**0.00886 ± 0.00171**	-	-
DWT-3D	CPU	0.39827 ± 0.04912	-	0.81744 ± 0.01047
	GPU	**0.08047 ± 0.04310**	-	-
	GPU-jit	0.08096 +- 0.00410	-	-

Table 1: Run-time comparisons for various implementations of the padded wavelet transformation from one to three dimensions. We compare transformations of $32 \cdot 10^6$ random values. Inputs are shaped as $\mathbb{R}^{32\times 10^6}$, $\mathbb{R}^{32\times 10^3\times 10^3}$ and $\mathbb{R}^{32\times 10^2\times 10^2\times 10^2}$ transformation run times are reported in seconds. All runs use a Daubechies five-wavelet. We report mean and standard deviations over 100 repetitions each. We explore the effect of Just In Time Compilation (jit) additionally to running on CPU and GPU. The separable (sep.) two-dimensional transform employs two single-dimensional transforms.

3. Comparison to Existing Work

We provide support for GPUs and gradient propagation for many functions, which used to be available only on Central Processing Units (sCPUs) without automatic differentiation-support. Additionally, we support boundary wavelets. The documentation lists all of `ptwts` features. Extensive unit testing ensures correct and `pywt`-consistent results.

3.1 Speed-tests

`ptwt` inherits GPU and jit support from PyTorch. All speed tests were run on a machine with an Intel Xeon W-2235 CPU @ 3.80GHz and an NVIDIA RTX A4000 Graphics card. Table 1 compares run times of Discrete Wavelet Transform (DWT) implementations for up to three dimensions. Adding GPU support yields significant speedups compared to Lee et al. (2019). Compared to the two-dimensional code presented in Cotter (2022), we observe state-of-the-art performance on GPU. Table 2 lists our measurements for the CWT-case. The input signal has dimensions of $\mathbb{R}^{32\times 10^3}$, with the first dimension the batch- and the second dimension the time dimension. All experiments use a Shannon wavelet. Here, we

		run-time [s]		
		ours	Cotter (2022)	Lee et al. (2019)
CWT	CPU	0.16029 ± 0.00925	-	0.94439 ± 0.01742
	GPU	0.01957 ± 0.01081	-	-
	GPU-jit	**0.01566 ± 0.00193**	-	-

Table 2: Run-time comparison for different implementations of the CWT. We report mean and standard deviations over 100 repetitions each.

see consistent computing-time reductions for each step from CPU, GPU, and jit. On CPUs, the switch to `ptwt` leads to a speedup of roughly a factor of four. Since we add the matrix form to the Python ecosystem, supplementary Figure 3 presents runtime measurements.

4. Conclusion

We presented selected features of the PyTorch Wavelet Toolbox. We extended the set of available methods on GPU by providing support for single and three-dimensional transforms in PyTorch. Where our tools overlap with alternative frameworks, we enable GPU and gradient support. Additionally, we allow Just In Time Compilation (jit). In terms of runtime, using `ptwt` leads to improvements in many cases. Last, but not least, our toolbox supports boundary wavelet computations for the first time in the Python world.

Acknowledgments

MW thanks Stefan Kesselheim for his feedback. MW acknowledges funding from the Bundesministerium für Bildung und Forschung under the BntrAInee and WestAI project grants. The authors gratefully acknowledge access to the Bender cluster hosted by the University of Bonn as well as the JUWELS Booster Partition at the Jülich Supercomputing Centre. CTH was funded under the Defense Advanced Research Projects Agency (DARPA) Automating Scientific Knowledge Extraction and Modeling program [HR00112220036].

Appendix A. Supplementary material

Acronyms

API Application Programming Interface

CPU Central Processing Unit

CWT Continuous Wavelet Transform

DWT Discrete Wavelet Transform

FWT Fast Wavelet Transform

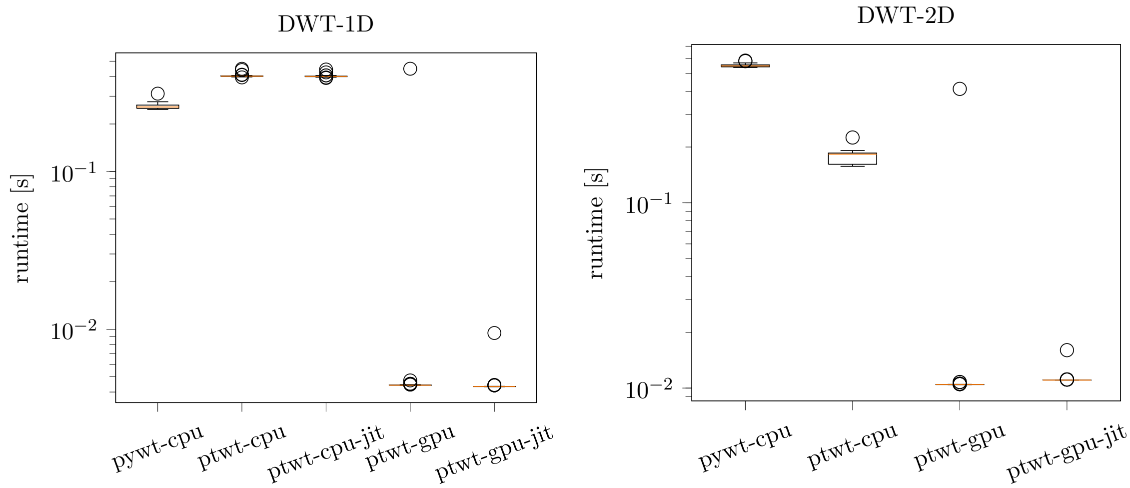

Figure 1: Run-time box-plots of our single dimensional (left) and two dimensional (right) padded DWT speed tests. The first run is typically significantly slower than subsequent runs. This behavior causes the outliers.

GPU Graphics Processing Unit

jit Just In Time Compilation

A.1 Code quality

We ensure code quality by running pytest, flake8, and mypy within an GitHub workflow. Nox ensures dependencies are installed correctly for all our tests. Pytest runs more than 4k test cases to ensure correct toolbox operation.

References

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs. `http://github.com/google/jax`, 2018.

Fergal Cotter. *Uses of Complex Wavelets in Deep Convolutional Neural Networks*. PhD thesis, University of Cambridge, 2020.

Fergal Cotter. 2d wavelet transforms in Pytorch. `https://github.com/fbcotter/pytorch_wavelets`, 2022.

Ingrid Daubechies. *Ten lectures on wavelets*. SIAM, 1992.

Konstantin Gasenzer and Moritz Wolter. Towards generalizing deep-audio fake detection networks. *arXiv preprint arXiv:2305.13033*, 2023.

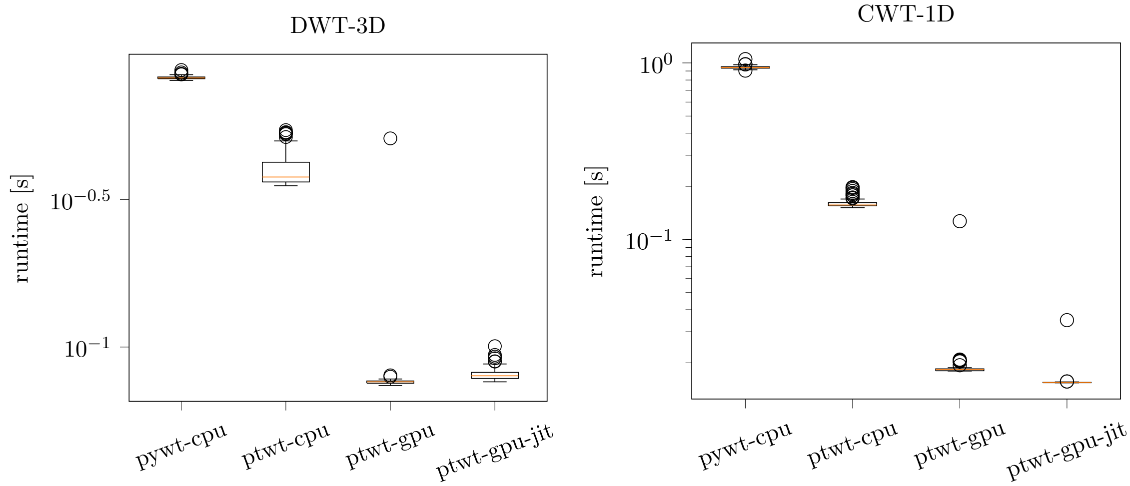

Figure 2: Run-time box-plots of the 3d-speed test (left) and for the continuous transform (right). The first run is typically significantly slower than subsequent runs. This behavior causes the outliers.

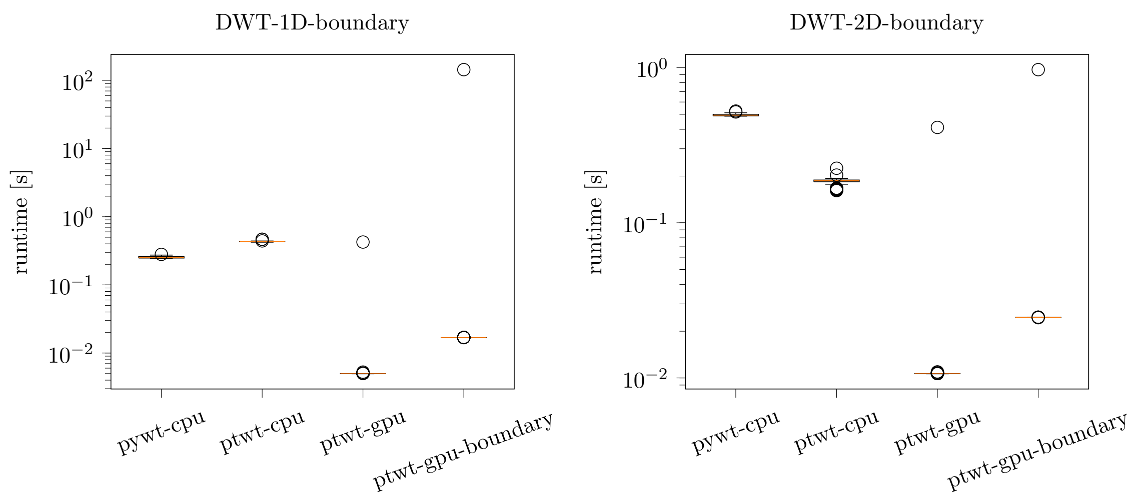

Figure 3: Run-time box-plots of the boundary wavelet code in one and two dimensions. The first run is typically significantly slower than subsequent runs. This behavior causes the outliers.

Wei Huang, Michelangelo Valsecchi, and Michael Multerer. Anisotropic multiresolution analyses for deep fake detection. *arXiv preprint arXiv:2210.14874*, 2022.

Arne Jensen and Anders la Cour-Harbo. *Ripples in mathematics: the discrete wavelet transform*. Springer Science & Business Media, 2001.

Gregory Lee, Ralf Gommers, Filip Waselewski, Kai Wohlfahrt, and Aaron O'Leary. PyWavelets: A Python package for wavelet analysis. *Journal of Open Source Software*, 4 (36):1237, 2019. URL `https://github.com/PyWavelets/pywt`.

Stéphane Mallat. *A Wavelet Tour of Signal Processing – The Sparse Way*. Academic Press, 3rd edition, 2008.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zach DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *31th International Conference on Artificial Neural Networks*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Daniel Recoskie. *Learning sparse orthogonal wavelet filters.* PhD thesis, University of Waterloo, 2018.

Gilbert Strang and Truong Nguyen. *Wavelets and filter banks.* SIAM, 1996.

Christopher Torrence and Gilbert P Compo. A practical guide to wavelet analysis. *Bulletin of the American Meteorological society*, 79(1):61–78, 1998.

Moritz Wolter and Jochen Garcke. Adaptive wavelet pooling for convolutional neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 1936–1944. PMLR, 2021.

Moritz Wolter, Shaohui Lin, and Angela Yao. Neural network compression via learnable wavelet transforms. In *29th International Conference on Artificial Neural Networks*, 2020.

Moritz Wolter, Felix Blanke, Raoul Heese, and Jochen Garcke. Wavelet-packets for deepfake image analysis and detection. *Machine Learning*, Special Issue of the ECML PKDD 2022 Journal Track:1–33, August 2022. ISSN 0885-6125. doi: https://doi.org/10.1007/s10994-022-06225-5. URL `https://rdcu.be/cUIRt`.