The Journal of Open Source Software

TorchMetrics - Measuring Reproducibility in PyTorch

Nicki Skafte Detlefsen[1,2], Jiri Borovec[1], Justus Schock[1,3], Ananya Harsh Jha[1], Teddy Koker[1], Luca Di Liello[4], Daniel Stancl[5], Changsheng Quan[6], Maxim Grechkin[7], and William Falcon[1,8]

1 Grid AI Labs **2** Technical University of Denmark **3** University Hospital Düsseldorf **4** University of Trento **5** Charles University **6** Zhejiang University **7** Independent Researcher **8** New York University

DOI: 10.21105/joss.04101

Software

- Review
- Repository
- Archive

Editor: Bita Hasheminezhad

Reviewers:

- @inpefess
- @richrobe
- @reneraab

Submitted: 17 December 2021
Published: 10 February 2022

Summary

A main problem with reproducing machine learning publications is the variance of metric implementations across papers. A lack of standardization leads to different behavior in mechanisms such as checkpointing, learning rate schedulers or early stopping, that will influence the reported results. For example, a complex metric such as Fréchet inception distance (FID) for synthetic image quality evaluation (Heusel et al., 2017) will differ based on the specific interpolation method used.

There have been a few attempts at tackling the reproducibility issues. Papers With Code (*Papers with Code*, n.d.) links research code with its corresponding paper. Similarly, arXiv (*Arxiv*, n.d.) recently added a code and data section that links both official and community code to papers. However, these methods rely on the paper code to be made publicly accessible which is not always possible. Our approach is to provide the de-facto reference implementation for metrics. This approach enables proprietary work to still be comparable as long as they've used our reference implementations.

We introduce TorchMetrics, a general-purpose metrics package that covers a wide variety of tasks and domains used in the machine learning community. TorchMetrics provides standard classification and regression metrics; and domain-specific metrics for audio, computer vision, natural language processing, and information retrieval. Our process for adding a new metric is as follows, first we integrate a well-tested and established third-party library. Once we've verified the implementations and written tests for them, we re-implement them in native PyTorch (Paszke et al., 2019) to enable hardware acceleration and remove any bottlenecks in inter-device transfer.

Statement of need

Currently, there is no standard, widely-adopted metrics library for native PyTorch. Some native PyTorch libraries support domain-specific metrics such as Transformers (Wolf et al., 2020) for calculating NLP-specific metrics. However, no library exists that covers multiple domains. PyTorch users, therefore, often rely on non-PyTorch packages such as Scikit-learn (Pedregosa et al., 2011) for computing even simple metrics such as accuracy, F1, or AUROC metrics.

However, while Scikit-learn is considered the gold standard for computing metrics in regression and classification, it relies on the core assumption that all predictions and targets are available simultaneously. This contradicts the typical workflow in a modern deep learning training/evaluation loop where data comes in batches. Therefore, the metric needs to be

calculated in an online fashion. It is important to note that, in general, it is not possible to calculate a global metric as its average or sum of the metric calculated per batch.

TorchMetrics solves this problem by introducing stateful metrics that can calculate metric values on a stream of data alongside the classical functional and stateless metrics provided by other packages like Scikit-learn. We do this with an effortless `update` and `compute` interface, well known from packages such as Keras (Chollet & others, 2015). The `update` function takes in a batch of predictions and targets and updates the internal state. For example, for a metric such as accuracy, the internal states are simply the number of correctly classified samples and the total observed number of samples. When all batches have been passed to the `update` method, the `compute` method can get the accumulated accuracy over all the batches. In addition to `update` and `compute`, each metric also has a `forward` method (as any other `torch.nn.Module`) that can be used to both get the metric on the current batch of data and accumulate global state. This enables the user to get fine-grained info about the metric on the individual batch and the global metric of how well their model is doing.

```python
# Minimal example showcasing the TorchMetrics interface
import torch
from torch import tensor, Tensor
# base class all modular metrics inherit from
from torchmetrics import Metric

class Accuracy(Metric):
    def __init__(self):
        super().__init__()
        # `self.add_state` defines the states of the metric
        #  that should be accumulated and will automatically
        #  be synchronized between devices
        self.add_state("correct", default=tensor(0), dist_reduce_fx="sum")
        self.add_state("total", default=tensor(0), dist_reduce_fx="sum")

    def update(self, preds: Tensor, target: Tensor) -> None:
        # update takes `preds` and `target` and accumulate the current
        # stream of data into the global states for later
        self.correct += torch.sum(preds == target)
        self.total += target.numel()

    def compute(self) -> Tensor:
        # compute takes the accumulated states
        # and returns the final metric value
        return self.correct / self.total
```

Another core feature of TorchMetrics is its ability to scale to multiple devices seamlessly. Modern deep learning models are often trained on hundreds of devices such as GPUs or TPUs (see (Liu et al., 2019; Zhai et al., 2021) for examples). This scale introduces the need to synchronize metrics across machines to get the correct value during training and evaluation. In distributed environments, TorchMetrics automatically accumulates across devices before reporting the calculated metric to the user.

In addition to stateful metrics (called modular metrics in TorchMetrics), we also support a functional interface that works similar to Scikit-learn. This interface provides simple Python functions that take as input PyTorch Tensors and return the corresponding metric as a PyTorch Tensor. These can be used when metrics are evaluated on single devices, and no accumulation is needed, making them very fast to compute.

TorchMetrics exhibits high test coverage on the various configurations, including all three major OS platforms (Linux, macOS, and Windows), and various Python, CUDA, and PyTorch

versions. We test both minimum and latest package requirements for all combinations of OS and Python versions and include additional tests for each PyTorch version from 1.3 up to future development versions. On every pull request and merge to master, we run a full test suite. All standard tests run on CPU. In addition, we run all tests on a multi-GPU setting which reflects realistic Deep Learning workloads. For usability, we have auto-generated HTML documentation (hosted at readthedocs) from the source code which updates in real-time with new merged pull requests.

TorchMetrics is released under the Apache 2.0 license. The source code is available at https://github.com/PyTorchLightning/metrics.

Acknowledgement

The TorchMetrics team thanks Thomas Chaton, Ethan Harris, Carlos Mocholí, Sean Narenthiran, Adrian Wälchli, and Ananth Subramaniam for contributing ideas, participating in discussions on API design, and completing Pull Request reviews. We also thank all of our open-source contributors for reporting and resolving issues with this package. We are grateful to the PyTorch Lightning team for their ongoing and dedicated support of this project, and Grid.ai for providing computing resources and cloud credits needed to run our Continuos Integrations.

References

Arxiv. (n.d.). https://arxiv.org/.

Chollet, F., & others. (2015). *Keras*. https://github.com/fchollet/keras; GitHub.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2017). GANs trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in Neural Information Processing Systems*, *30*, 6629–6640. https://proceedings.neurips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). *RoBERTa: A robustly optimized BERT pretraining approach*. http://arxiv.org/abs/1907.11692

Papers with code. (n.d.). https://paperswithcode.com/.

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 H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P. von, Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., … Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods*

in Natural Language Processing: System Demonstrations, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

Zhai, X., Kolesnikov, A., Houlsby, N., & Beyer, L. (2021). *Scaling vision transformers*. http://arxiv.org/abs/2106.04560