

Supplementary Material

cblearn: Comparison-based Machine Learning in Python

David-Elias Küstle and Ulrike von Luxburg
University of Tübingen and Tübingen AI Center, Germany

22 September 2023

Empirical evaluation

We generated embeddings of comparison-based datasets to measure runtime and triplet error as a small empirical evaluation of our ordinal embedding implementations. We compared various CPU and GPU implementations in **cblearn** with third-party implementations in *R* (see Terada and Luxburg 2014), and *MATLAB* (Maaten and Weinberger 2012). In contrast to synthetic benchmarks (e.g., Vankadara et al. 2021), we used real-world datasets that can be accessed and converted to triplets through **cblearn**. The embeddings were arbitrarily chosen to be 2D. Every algorithm runs once per dataset on a compute node (8 CPU cores; 96GB RAM; NVIDIA RTX 2080ti) with a run-time limit of 24 hours. Some runs failed by exceeding those constraints: our FORTE implementation failed due to an “out of memory” error on the **imagenet-v2** dataset. The *MATLAB* implementation of tSTE timed out on **things** and **imagenet-v2** datasets. The run of the *R* SOE implementation on the **imagenet-v2** dataset failed by an “unsupported long vector” error caused by the large size of the requested embedding.

The benchmarking scripts and results are publicly available¹.

Is there a “best” estimator?

Comparing the ordinal embedding estimators in **cblearn**, SOE, CKL, GNMDS, and tSTE were performing about equally well in both runtime and accuracy (Figure 1). The GPU implementations are slower on the tested datasets and noticeably less accurate for SOE and GNMDS.

¹<https://github.com/cblearn/cblearn-benchmark>

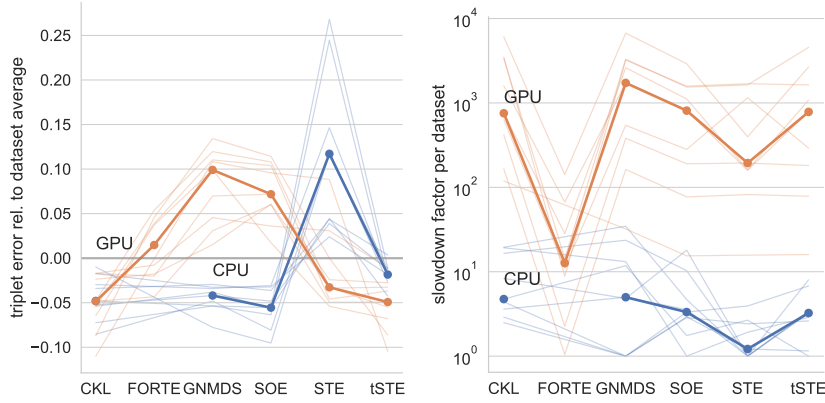


Figure 1: The triplet error and runtime per estimator and dataset relative to the mean error or the fastest run. Thin lines show runs on the different datasets; the thick lines indicate the respective median. Except for STE, all CPU algorithms can embed the triplets similarly well. There are just minor differences in the runtime of the CPU implementations. The GPU implementations are usually significantly slower on the data sets used.

When should GPU implementations be preferred?

Regarding accuracy and runtime, our GPU implementations using the `torch` backend could not outperform the CPU pendants using the `scipy` backend on the tested datasets. However, Figure 1 shows the GPU runtime grows slower with the number of triplets, such that they potentially outperform CPU implementations with large datasets of 10^7 triplets and more. Sometimes, the `torch` implementations show the best accuracy.

We could think of various explanations for the speed disadvantage of our `torch` implementations. On the one hand, it may be due to the overhead of converting between `numpy` and `torch` and calculating the gradient (AutoGrad). On the other hand, it can also be due to the optimizer or the selected hyperparameters. To get a first impression of these factors, we have built minimal examples of the CKL algorithm (Tamuz et al. 2011) and estimated 2D embeddings of the Vogue Cover dataset (Heikinheimo and Ukkonen 2013). Figure 3 shows a standard laptop’s runtimes and triplet accuracies. The small markers show runs with different initializations, and the bold markers show the respective median performance. The CKL implementation of `cblearn` is slightly slower than the minimal version, probably due to data validation and conversion overheads. If the gradient is not provided directly but calculated automatically with PyTorch’s AutoGrad functions, the minimal example runs multiple times slower. The most severe impact has been to change the optimization algorithm to stochastic optimization (*Adam*, $lr=10$). However, following the results in previous sections, it can be

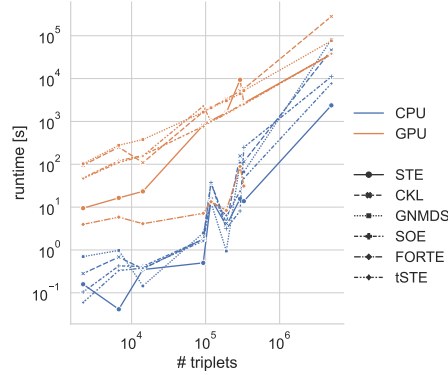


Figure 2: The runtime increases almost linearly with the number of triplets. However, GPU implementations have a flatter slope and thus can compensate for the initial time overhead on large datasets.

assumed that this overhead is compensated for by increasing the dataset size.

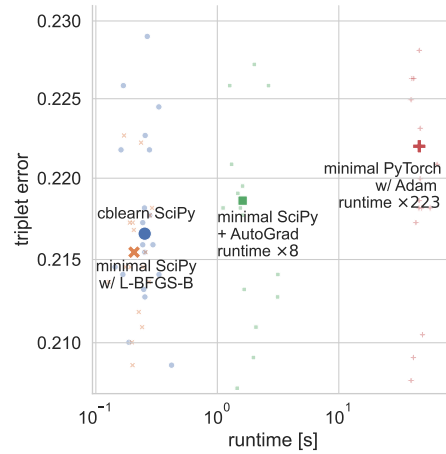


Figure 3: The runtime and error for different optimization methods in minimal CKL implementations. `cblearn`'s CKL implementation is shown for reference.

Another challenge for stochastic optimizers like *Adam* (Kingma and Ba 2015) is their sensitivity to hyperparameter choices. This sensitivity is demonstrated in Figure 4, where the learning rate of Adam is varied for the toy example. Likewise, tuning the optimizer parameters could improve the performance of the `torch` ordinal embedding implementations.

Besides all discussions about runtime and accuracy, the `torch` backend provides benefits for maintaining and expanding the library. It uses PyTorch's automatic

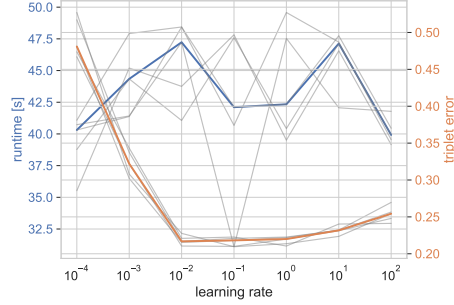


Figure 4: The runtime and error for different learning rates of the Adam optimizer in a minimal example with CKL estimating a 2D embedding of 60 objects.

differentiation (Paszke et al. 2019) so that the loss gradient does not have to be explicitly defined, and new algorithms can be implemented very quickly.

How does cblearn compare to other implementations?

In a small comparison, our implementations run multiple times faster with approximately the same accuracy as reference implementations (Figure 5). We compared our CPU implementations of SOE the corresponding reference implementations in *R*, *loe* (Terada and Luxburg 2014), and our implementation of CKL, GNMDS, STE, tSTE with the *MATLAB* of Maaten and Weinberger (2012). This comparison is not exhaustive, but it shows that our implementations are competitive with the reference implementations in terms of accuracy and runtime. Of course, we cannot separate the factors of algorithm implementation and runtime environment.

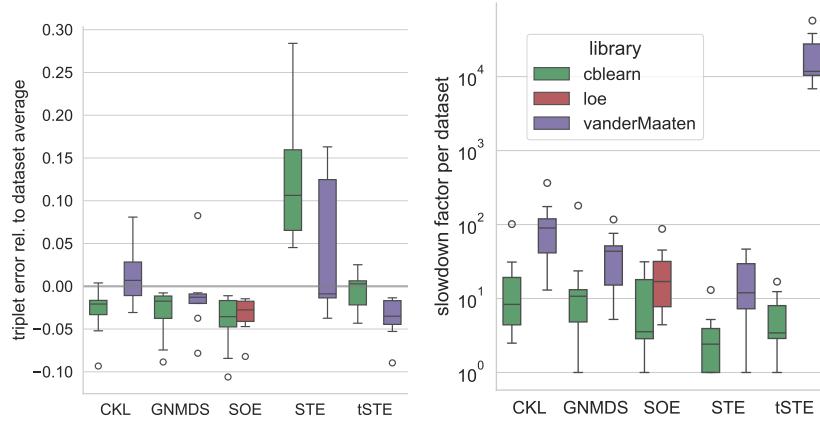
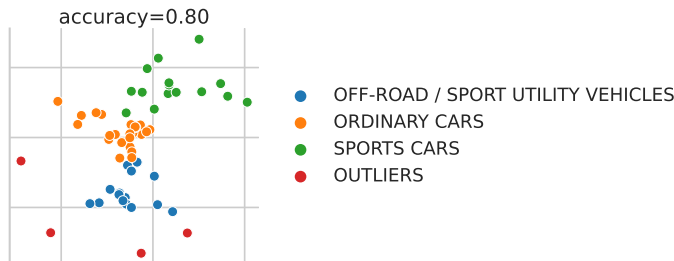


Figure 5: The triplet error and runtime per estimator relative to the mean error and the fastest run on each dataset. Thin lines show runs on the different datasets; the thick lines indicate the respective median. The triplet error is approximately similar for all implementations but STE. For all algorithms, ‘cblearn’ provides the fastest implementation.

Code example

```
from cblearn import datasets, preprocessing, embedding
from sklearn.model_selection import cross_val_score
import seaborn as sns; sns.set_theme("poster", "whitegrid")

cars = datasets.fetch_car_similarity()
triplets = preprocessing.triplets_from_mostcentral(cars.triplet, cars.response)
accuracy = cross_val_score(embedding.SOE(n_components=2), triplets, cv=5).mean()
embedding = embedding.SOE(n_components=2).fit_transform(triplets)
fig = sns.relplot(x=embedding[:, 0], y=embedding[:, 1],
                  hue=cars.class_name[cars.class_id])
fig.set(title=f"accuracy={accuracy:.2f}", xticklabels=[], yticklabels=[])
fig.tight_layout(); fig.savefig("images/car_example.pdf")
```



References

- Heikinheimo, Hannes, and Antti Ukkonen. 2013. “The Crowd-Median Algorithm.” In *Proceedings of the Aaai Conference on Human Computation and Crowdsourcing*, 1:69–77.
- Kingma, Diederik P., and Jimmy Ba. 2015. “Adam: A Method for Stochastic Optimization.” In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, ca, Usa, May 7-9, 2015, Conference Track Proceedings*, edited by Yoshua Bengio and Yann LeCun. <http://arxiv.org/abs/1412.6980>.
- Maaten, Laurens van der, and Kilian Weinberger. 2012. “Stochastic Triplet Embedding.” In *International Workshop on Machine Learning for Signal Processing*, 1–6. <https://doi.org/10.1109/MLSP.2012.6349720>.

- Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, et al. 2019. “Pytorch: An Imperative Style, High-Performance Deep Learning Library.” *Advances in Neural Information Processing Systems* 32.
- Tamuz, Omer, Ce Liu, Serge Belongie, Ohad Shamir, and Adam Tauman Kalai. 2011. “Adaptively Learning the Crowd Kernel.” In *Proceedings of the 28th International Conference on International Conference on Machine Learning (Icml)*.
- Terada, Yoshikazu, and Ulrike Luxburg. 2014. “Local Ordinal Embedding.” In *International Conference on Machine Learning (Icml)*.
- Vankadara, Leena Chennuru, Siavash Haghiri, Michael Lohaus, Faiz Ul Wahab, and Ulrike Luxburg. 2021. “Insights into Ordinal Embedding Algorithms: A Systematic Evaluation.” *arXiv:1912.01666 [Cs, Stat]*. <https://doi.org/10.48550/arXiv.1912.01666>.