openTSNE: a modular Python library for t-SNE dimensionality reduction and embedding

Pavlin G. Poličar¹,*, Martin Stražar¹, and Blaž Zupan¹,²

¹Faculty of Computer and Information Science, University of Ljubljana, SI-1000 Ljubljana, Slovenia,
²Department of Molecular and Human Genetics, Baylor College of Medicine, Houston TX 77030, U.S.A.

* To whom correspondence should be addressed.

Abstract

Summary: Point-based visualisations of large, multi-dimensional data from molecular biology can reveal meaningful clusters. One of the most popular techniques to construct such visualisations is t-distributed stochastic neighbor embedding (t-SNE), for which a number of extensions have recently been proposed to address issues of scalability and the quality of the resulting visualisations. We introduce openTSNE, a modular Python library that implements the core t-SNE algorithm and its extensions. The library is orders of magnitude faster than existing popular implementations, including those from scikit-learn. Unique to openTSNE is also the mapping of new data to existing embeddings, which can surprisingly assist in solving batch effects.

Availability: openTSNE is available at https://github.com/pavlin-policar/openTSNE.

Contact: pavlin.policar@fri.uni-lj.si, blaz.zupan@fri.uni-lj.si

We introduce openTSNE, a comprehensive Python library that implements t-SNE and all its recently proposed extensions. The library is compatible with the Python data science ecosystem (e.g., numpy, sklearn, scanpy). Its modular design encourages extendibility and experimentation with various settings and changes in the analysis pipeline. For example, the following code uses multiscale similarity kernels to construct the embedding from Fig. 1.b.

```python
adata = anndata.read_h5ad("macsko_2015.h5ad")
affinities = openTSNE.affinity.Multiscale(adata.obsm["pca"], perplexities=[50, 500], metric="cosine")
init = openTSNE.initialization.pca(adata.obsm["pca"])
embedding = TSNEEmbedding(init, affinities)
embedding.optimize(n_iter=250, exaggeration=12, momentum=0.5, inplace=True)
embedding.optimize(n_iter=750, momentum=0.8, inplace=True)
```

Here, we first read the data, define the affinity model based on two Gaussian kernels with varying perplexity, use a PCA-based initialization, and run the typical two-stage t-SNE optimization. Notice that the code for the standard t-SNE used for Fig. 1.a is similar but uses only a single kernel (perplexities=[30]).

The proposed openTSNE library is currently the only Python t-SNE implementation that supports adding new samples into constructed embedding. For example, we can reuse the embedding created above to map new data into existing embedding space in the following code.

```python
data = anndata.read_h5ad("shekhar_2016.h5ad")
data, genes = find_shared_genes(adata, data)
genes = select_genes(adata.X, n=1000)
new_embedding = embedding.transform(data[:, genes].X)
```

which loads and prepare the new data, defines the affinity model, and computes the embeddings. openTSNE embeds new data points independently of one another, without changing the reference embedding. The example of combining two data sets with mouse retina cells is shown on Fig. 1.c,
where the cells from the secondary data set are matched to the cells in the reference embedding. This procedure can therefore be used to handle batch effects (10), a key obstacle in molecular biology when dealing with the data from different sources (11). The code used to plot the embeddings is not shown for brevity, but is, together with other examples, available on openTSNE's GitHub page.

Our Python implementation introduces computational overhead: openTSNE is about 25% slower than FIt-SNE (7), a recent t-SNE implementation in C++. However, openTSNE is still orders of magnitude faster than other Python implementations, including those from scikit-learn and MulticoreTSNE (see Benchmarks in openTSNE documentation on GitHub). An example data set with 200,000 cells is processed in more than 90 minutes with scikit-learn and in less than 4 minutes with openTSNE. The framework includes controlled execution (callback-based progress monitoring and control), making it suitable for interactive data exploration environments such as Orange (12). A pure Python implementation offers distinct advantages that include integration with Python’s rich data science infrastructure and ease of installation through PyPI and conda.

**Funding and Acknowledgements**
The support for this work was provided by the Slovenian Research Agency Program Grant P2-0209, and by European Regional Development Fund and the Slovenian Ministry of Education through BioPharm project. We want to thank George Linderman and Dmitry Kobak for helpful discussions on extensions of t-SNE.

**Bibliography**