OXFORD

Data and text mining

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

Pavlin G. Poličar [1,*] **Martin Stražar** [1] **and Blaž Zupan** [1, 2]

[1] Faculty of Computer and Information Science, University of Ljubljana, SI-1000 Ljubljana, Slovenia,
[2] Department 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 structures and 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 speed and quality of the resulting visualisations. We propose 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 an existing embedding, which can surprisingly assist in solving batch effect problems.
Availability: openTSNE is available at `https://github.com/pavlin-policar/openTSNE`.
Contact: pavlin.policar@fri.uni-lj.si

The abundance of high-dimensional data sets in molecular biology calls for techniques for dimensionality reduction, and in particular for methods that can help in the construction of data visualizations. Popular approaches for dimensionality reduction include principal component analysis, multidimensional scaling, and uniform manifold approximation and projections (). Among these, t-distributed stochastic neighbor embedding (t-SNE) (Maaten and Hinton, 2008; **?**) lately received much attention as it can address high volumes of data and reveal clustering structure. Most of the recent reports on single-cell gene expression data would start with an overview of the cell landscape, where t-SNE embeds high-dimensional expression profiles into two dimensional space (Macosko *et al.*, 2015; Shekhar *et al.*, 2016; **?**). Fig. 1.a presents an example of such an embedding.

Despite its utility, t-SNE has lately been criticized for insufficient speed when addressing huge data sets, lack of global organization *t-SNE focuses on local clusters that are then arbitrarily scattered in the low-dimensional space* and absence of theoretically-founded implementations to map new data into an existing embedding. Most of these shortcomings, however, were recently addressed (Ding *et al.*, 2018; Becht *et al.*, 2019). Linderman *et al.* (2019) sped-up the method through interpolation-based approximation, reducing the time complexity to be merely linear in

the number of data items. Kobak and Berens (2018) proposed several techniques to improve global positioning, including estimating similarities with a mixture of Gaussian kernels. While no current popular software library supports mapping of new data into reference embedding, Maaten (2009) proposed a related approach with parametric mapping using neural networks.

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

```
adata = anndata.read_h5ad("macosko_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,

```
new_data = anndata.read_h5ad("shekhar_2016.h5ad")
adata, new_data = find_shared_genes(adata, new_data)
gene_mask = select_genes(adata.X, n=1000)
embedding.affinities =
    affinity.PerplexityBasedNN(adata[:, gene_mask].X,
    perplexity=30, metric="cosine")
new_embedding = embedding.transform(new_data[:,
    gene_mask].X)
```

which loads and prepare the new data, defines the affinity model, and computes the embeddings. `openTSNE` embeds new data one data instance at the time, 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 of the secondary data set get matched to the cells in the reference embedding. This procedure can therefore be used for handling batch effects **?**, a substantial problem in molecular biology when dealing with the data from different sources **?**. The code for plotting of the data 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` (Linderman *et al.*, 2019), 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 sets with 200,000 cells is processed in more than 90 minutes with scikit-learn and 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 **?**. 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

References

Becht, E. et al (2019). Dimensionality reduction for visualizing single-cell data using umap. *Nature Biotechnology*, **37**(1), 38.

Ding, J. et al (2018). Interpretable dimensionality reduction of single cell transcriptome data with deep generative models. *Nature Communications*, **9**(1), 2002.

Kobak, D. and Berens, P. (2018). The art of using t-sne for single-cell transcriptomics. *bioRxiv*, page 453449.

Linderman, G.C. et al (2019). Fast interpolation-based t-sne for improved visualization of single-cell rna-seq data. *Nature methods*, **16**(3), 243.

Maaten, L. (2009). Learning a parametric embedding by preserving local structure. In *Artificial Intelligence and Statistics*, pages 384–391.

Maaten, L.v.d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, **9**(Nov), 2579–2605.

Macosko, E.Z. et al (2015). Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell*, **161**(5), 1202–1214.

Shekhar, K. et al (2016). Comprehensive Classification of Retinal Bipolar Neurons by Single-Cell Transcriptomics. *Cell*, **166**(5), 1308–1323.
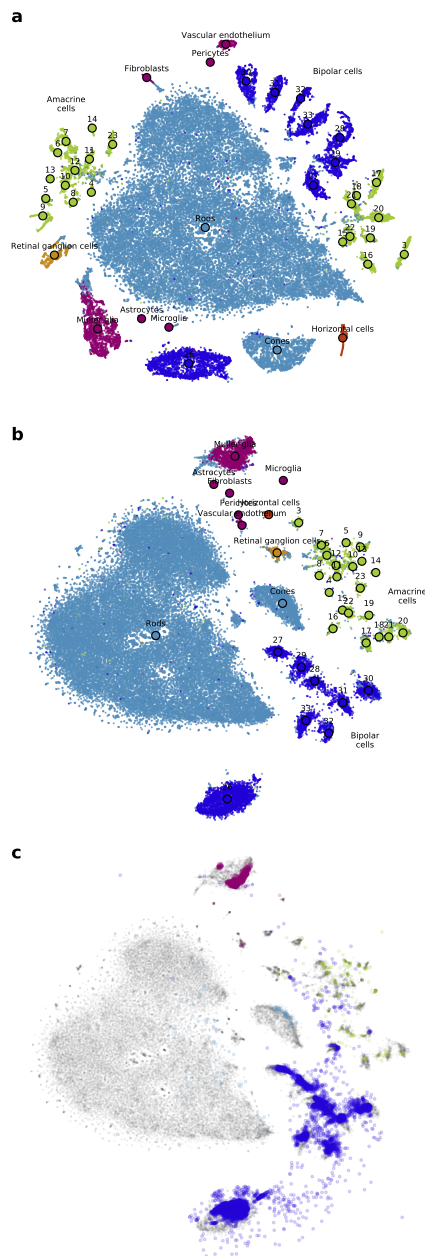
Fig. 1. Example application of `openTSNE` on mouse retina cells from Macosko et al. (2015) and Shekhar et al. (2016). a) Standard t-SNE embedding using random initialization and perplexity of 30. b) t-SNE embedding using multi-scale affinities leads to better global cluster organization. Cluster annotations in both a) and b) are from Macosko et al. c) Embedding of independent mouse retina cells from Shekhar et al. (points in color) to a reference t-SNE visualization of data from Macosko et al. (points in gray) places cells in the clusters that match classifications from original publications and alleviates batch effects.