Journal of Statistical Software
MMMMMM YYYY, Volume VV, Issue II. doi: 10.18637/jss.v000.i00

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

Pavlin G. Poličar
Faculty of Computer and Information Science
University of Ljubljana
Ljubljana, Slovenia

Martin Stražar
Broad Institute of MIT and Harvard
Cambridge, MA, USA

Blaž Zupan
Faculty of Computer and Information Science
University of Ljubljana
Ljubljana, Slovenia

Abstract

One of the most popular techniques for constructing visualizations of large, high-dimensional data sets is *t*-distributed stochastic neighbor embedding (t-SNE). Recently, several extensions have been proposed to address issues of scalability and the quality of the resulting visualizations. We introduce **openTSNE**, a modular Python library that implements the core t-SNE algorithm and its many extensions. The library is fast and scales up to data sets containing millions of data points. **openTSNE** is distributed under the BSD-3-Clause License and its source code is publicly available at https://github.com/pavlin-policar/openTSNE.

Keywords: t-SNE, embedding, visualization, dimensionality reduction, Python.

1. Introduction

The ever-growing volumes of high-dimensional data sets in machine learning calls for efficient dimensionality reduction techniques and their implementations, which can produce informative data visualizations. Popular approaches include principal component analysis, multidimensional scaling, t-distributed stochastic neighbor embedding (t-SNE) (Van Der Maaten and Hinton 2008), and uniform manifold approximation and projections (UMAP) (McInnes, Healy, and Melville 2018). Among these, t-SNE has received much attention as it can address high volumes of data and reveal the underlying data structure. For instance, t-SNE is widely used in the bioinformatics community in areas such as single-cell transcriptomics (Macosko, Basu, Satija, Nemesh, Shekhar, Goldman, Tirosh, Bialas, Kamitaki, Martersteck *et al.* 2015; Cao, Spielmann, Qiu, Huang, Ibrahim, Hill, Zhang, Mundlos, Christiansen, Steemers *et al.* 2019; Tasic, Yao, Graybuck, Smith, Nguyen, Bertagnolli, Goldy, Garren, Economo, Viswanathan *et al.* 2018), human genetics (Hirata, Hosomichi, Sakaue, Kanai, Nakaoka, Ishigaki, Suzuki, Akiyama, Kishikawa, Ogawa *et al.* 2019), metagenomic assembly (Beaulaurier, Zhu, Deikus, Mogno, Zhang, Davis-Richardson, Canepa, Triplett, Faith, Sebra *et al.* 2018), the spatial organization of microbial communities (Sheth, Li, Jiang, Sims, Leong, and Wang 2019), and metabolomics (Tkachev, Stepanova, Zhang, Khrameeva, Zubkov, Giavalisco, and Khaitovich 2019). Reports on single-cell gene expression data, our running example, often start with an overview of the cell landscape, where t-SNE is used to embed high-dimensional expression profiles into a two-dimensional space. Figs. 1.a and 1.b show two such embeddings.

Despite its utility, t-SNE has often been criticized for its limited scalability, lack of global organization – t-SNE identifies well-defined clusters that may be arbitrarily scattered throughout the low-dimensional space – and the absence of theoretically-founded methods to map new data into existing embeddings (Ding, Condon, and Shah 2018; Becht, McInnes, Healy, Dutertre, Kwok, Ng, Ginhoux, and Newell 2019). Most of these shortcomings have recently been addressed. Linderman, Rachh, Hoskins, Steinerberger, and Kluger (2019) developed FIt-SNE, an efficient approximation scheme which massively improves the scalability of t-SNE, achieving linear time complexity in the number of samples. Kobak and Berens (2019) proposed several techniques to improve global cluster coherence, including estimating similarities with a mixture of Gaussian kernels. In our previous work, we introduced a principled approach for embedding new samples into existing visualizations (Poličar, Stražar, and Zupan 2019).

However, easy access to efficient implementations almost universally correlates with the widespread adoption of novel computational techniques. Despite the many theoretical advancements, most popular t-SNE libraries have been slow to incorporate them into their implementations. In this paper, we present **openTSNE**, our open-source Python implementation of t-SNE and its many recently proposed extensions. **openTSNE** is easy to install and includes precompiled binaries available through popular Python package managers. It provides a familiar API, making it suitable as a drop-in replacement for existing t-SNE implementations.

2. Methods

We first introduce the relevant notation and briefly review the core t-SNE algorithm in Sec. 2.1. We then address the criticisms of scalability (Sec. 2.2), the ability to add new samples to existing embeddings (Sec. 2.3), and improvements that improve the global consistency of resulting embeddings (Sec. 2.4).

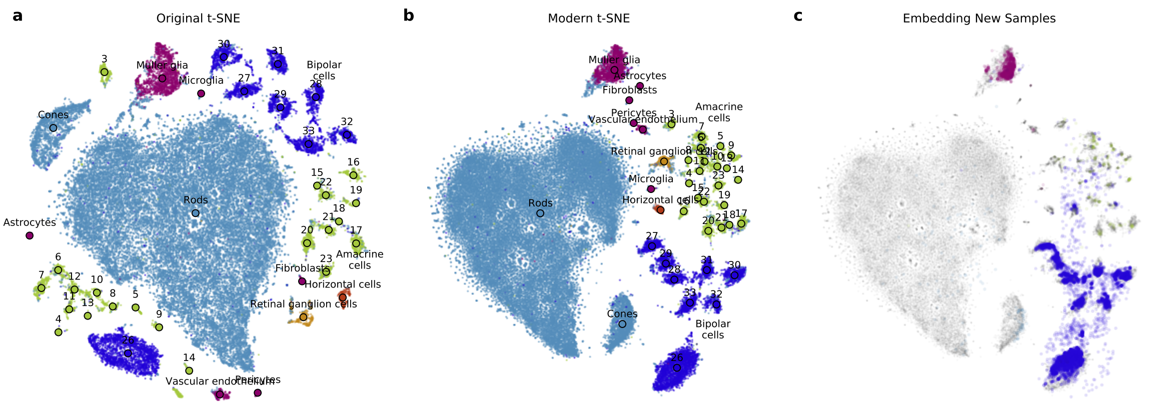

Figure 1: We use **openTSNE** to generate three t-SNE embeddings and demonstrate recent theoretical advances. The data in **(a)** and **(b)** represent 44,808 single-cell gene-expression profiles of mouse retinal cells from Macosko *et al.* (2015). The data in **(c)** additionally contains 27,499 expression profiles from mouse retinal cells from Shekhar *et al.* (2016). **(a)** We construct a t-SNE embedding following the parameter choices from the original publication by Van Der Maaten and Hinton (2008). The visualization shows no preservation of the global organization of clusters, resulting from random initialization and an affinity model focused on preserving local neighborhoods. **(b)** A modern t-SNE embedding, utilizing the latest theoretical advances and practical recommendations constructed using a multi-scale affinity model, preserving both short-range and long-range interactions between data points and initialized so that the global layout is as meaningful as possible. Unlike in **(a)**, the green and blue clusters representing different sub-types of amacrine and bipolar cells are now localized to the same regions of the space, indicating a higher level of similarity than to other cell types. The embedding in **(c)** shows how existing t-SNE reference atlases can be used to place new samples into existing embeddings. The positions of new data points correspond to cell types from the reference atlas.

2.1. t-distributed stochastic neighbor embedding

t-distributed stochastic neighbor embedding (t-SNE) is a non-linear dimensionality reduction method that to aims to find a low-dimensional embedding where neighborhoods are preserved. More formally, given a multi-dimensional data set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \in \mathbb{R}^D$ where N is the number of data points in the data set, t-SNE aims to find a low dimensional embedding $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\} \in \mathbb{R}^d$ where $d \ll D$, such that if points $\mathbf{x}_i$ and $\mathbf{x}_j$ are close in the high-dimensional space, their corresponding embeddings $\mathbf{y}_i$ and $\mathbf{y}_j$ are also close. Since t-SNE is primarily used as a visualization tool, d is typically set to two. The similarity between two data points in the high-dimensional space is defined using the Gaussian kernel

$$p_{j|i} = \frac{\exp\left(-\frac{1}{2}\mathcal{D}(\mathbf{x}_i, \mathbf{x}_j)/\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\frac{1}{2}\mathcal{D}(\mathbf{x}_i, \mathbf{x}_k)/\sigma_i^2\right)}, \quad p_{i|i} = 0$$

where $\mathcal{D}$ is some distance measure. The values $p_{j|i}$ are then symmetrized to

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}. \tag{1}$$

The bandwidth σ_i of each Gaussian kernel is selected such that the perplexity of the distribution matches a user-specified parameter value

$$\text{Perplexity} = 2^{H(P_i)}$$

where $H(P_i)$ is the Shannon entropy of P_i,

$$H(P_i) = -\sum_i p_{j|i} \log_2(p_{j|i}).$$

This enables t-SNE to adapt to the varying density of the data in the multi-dimensional space. The perplexity can be interpreted as as the continuous analogue to the number of nearest neighbors to which the distances will be preserved.

The similarity between points $\mathbf{y}_i$ and $\mathbf{y}_j$ in the embedding space is defined using the *t*-distribution with a single degree of freedom (Cauchy kernel)

$$q_{ij} = \frac{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1}}{\sum_{k \neq l} \left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)^{-1}}, \quad q_{ii} = 0. \tag{2}$$

We use the Kullback-Leibler (KL) divergence to measure the agreement between the two distributions $\mathbf{P}$ and $\mathbf{Q}$

$$C = \text{KL}(\mathbf{P} \mid\mid \mathbf{Q}) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{3}$$

The objective is to find embeddings $\mathbf{Y}$ that minimize the KL divergence. The corresponding gradient takes the form

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} \left(p_{ij} - q_{ij}\right) \left(\mathbf{y}_i - \mathbf{y}_j\right) w_{ij}, \tag{4}$$

where $w_{ij} = \left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1}$ and represents the unnormalized q_{ij}.

Optimization is performed using batch gradient descent using the delta-bar-delta update rule (Jacobs 1988). The t-SNE optimization proceedure consists of two phases: in the first *early exaggeration* phase, the attractive forces between data points are increased by some factor ρ, typically set to 12, so that points in the embedding can more easily move throughout the space and settle near their respective neighbors. In the second phase of the optimization, the attractive forces are then reverted to their original values with $\rho = 1$.

Belkina, Ciccolella, Anno, Halpert, Spidlen, and Snyder-Cappione (2019) later found that convergence can be sped up by increasing the learning rate from the standard $\eta = 200$ to $\eta = N/12$. As a side-effect, embeddings converge faster, and the number of iterations can be lowered from the typical 1000 to 750, decreasing the overall runtime. Modern t-SNE implementations, including **FIt-SNE** and **openTSNE**, have adopted this convention.

2.2. Efficient Approximation Schemes

A direct evaluation of t-SNE gradients requires $\mathcal{O}(N^2)$ operations, which makes its application impractical to any reasonably-sized data set and highlights the need for the development of efficient approximation schemes. Van Der Maaten (2014) observed that the t-SNE gradient can be cast as an N-body problem where data points represent particles that attract and repel each other. The gradient from Eqn. (4) can be rewritten as

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4\left[\underbrace{\sum_{j\neq i} p_{ij} q_{ij} Z \left(\mathbf{y}_i - \mathbf{y}_j\right)}_{\text{attractive forces}} - \underbrace{\sum_{j\neq i} q_{ij}^2 Z \left(\mathbf{y}_i - \mathbf{y}_j\right)}_{\text{repulsive forces}}\right],$$

where $Z = \sum_{k\neq l} \left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)^{-1}$. We can view this equation as a particle simulation, where the two terms represent the attractive and repulsive forces between individual particles. Each term lends itself to efficient approximations, enabling us to reduce the time complexity of t-SNE to $\mathcal{O}(N)$. This allows modern t-SNE implementations to be leveraged for the visulization of data sets containing up to millions of data points.

Attractive Forces

Van Der Maaten (2014) observed that evaluating the attractive forces between all pairs of data points is excessive, and that considering only a handful of nearest neighbors at each point are sufficient to obtain a good approximation. Therefore, instead of calculating all pairwise interactions, it is sufficient to only find and evaluate the attraction to a certain number of k nearest neighbors. Using tree-based nearest-neighbor search methods, Van Der Maaten (2014) reduced the time complexity to $\mathcal{O}(N \log N)$. Linderman *et al.* (2019) further realized that, qualitatively, embeddings are visually indistinguishable when using only *approximate* nearest neighbors, further reducing time complexity to $\mathcal{O}(N)$.

Repulsive Forces

The repulsive term can similarily be approximated, motivated by methods from particle simulations. Van Der Maaten (2014) proposed an approach based on N-body simulations and used a space-partitioning Barnes-Hut tree approach to approximate the interaction between data points. This reduces the time complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. More recently, Linderman *et al.* (2019) proposed an alternative approach, FIt-SNE, based on non-uniform convolutions which further reduces the time complexity to $\mathcal{O}(N)$.

2.3. Embedding New Samples

t-SNE is non-parametric and does not define an explicit mapping from the high-dimensional space to the embedding space. Therefore embeddings of new data points need to be found through the use of optimization techniques (Poličar *et al.* 2019). When adding new data points to an existing, reference embedding, the reference data points are fixed in place while new data points are allowed to find their respective positions. The optimization remains the same as in standard t-SNE with only slight modifications to p_{ij} and q_{ij}

$$p_{j|i} = \frac{\exp\left(-\frac{1}{2}\mathcal{D}(\mathbf{x}_i, \mathbf{v}_j)/\sigma_i^2\right)}{\sum_i \exp\left(-\frac{1}{2}\mathcal{D}(\mathbf{x}_i, \mathbf{v}_j)/\sigma_i^2\right)}, \qquad q_{j|i} = \frac{\left(1 + ||\mathbf{y}_i - \mathbf{w}_j||^2\right)^{-1}}{\sum_i \left(1 + ||\mathbf{y}_i - \mathbf{w}_j||^2\right)^{-1}},$$

where $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M\} \in \mathbb{R}^D$ where M is the number of samples in the new data set and $\mathbf{W} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M\} \in \mathbb{R}^d$. Additionally, we omit the symmetrization step in Eqn. (1).

Plugging these terms into Eqn. (3), we obtain the following gradient

$$\frac{\partial C}{\partial \mathbf{w}_j} = 2 \sum_i \left(p_{j|i} - q_{j|i} \right) (\mathbf{y}_i - \mathbf{w}_j) \left(1 + ||\mathbf{y}_i - \mathbf{w}_j||^2 \right)^{-1}.$$

Similarly to standard t-SNE, a direct calculation of gradients takes $\mathcal{O}(N \cdot M)$ time, but it is straightforward to adapt the Barnes-Hut and FIt-SNE approximation schemes, reducing the time complexity to $\mathcal{O}(M \log N)$ and $\mathcal{O}(M)$, respectively. Special care must be taken to tune the learning rate during optimization as the existing values $\eta = 200$ and $\eta = N/12$ result in unstable optimization.

2.4. Improving the Global Structure of Embeddings

One of t-SNE's criticisms is that it fails to preserve long-range distances but instead focuses on capturing the local structure of the data manifold (Becht *et al.* 2019). Recently, several approaches have been proposed to improve the global organization of the resulting embeddings.

Kobak and Linderman (2021) showed that the initialization of force-directed layout algorithms, such as t-SNE, UMAP (McInnes *et al.* 2018), and ForceAtlas2 (Jacomy, Venturini, Heymann, and Bastian 2014), largely dictates the global consistency of resulting embeddings. When initialized randomly, clusters are often arbitrarily scattered in the embedding space. However, when using initazation schemes based on PCA or spectral embeddings, the clusters identified in the resulting embedding are typically grouped in a globally coherent manner.

In standard t-SNE, distances between data points are converted to similarities through the use of Gaussian kernels. The perplexity of these kernels governs the sizes of the local neighborhoods which are to be preserved. One easy way to improve the global consistency of resulting embeddings is to increase the sizes of these neighborhoods. This can be achieved by increasing the perplexity parameter. However, this often comes at the price of local structure. Kobak and Berens (2019) propose to instead use mixtures of Gaussians to better preserve short-range, as well as long-range distances and, thus, achieve a better trade-off between local and global structure.

Embeddings produced by standard t-SNE often makes use all available space and separates clusters by only thin boundaries. When working with large data sets, this often obscures the global relationships between clusters as all neighboring groups appear equidistant from one another. Other dimensionality reduction methods, such as UMAP or ForceAtlas2, produce embeddings where clusters appear more compact and the white-space separating the clusters may be interpreted as a loose measure of distance. Recently, Böhm, Berens, and Kobak (2020) showed that the exaggeration factor ρ could be used to produce layouts more similar to UMAP and ForceAtlas2. By incorporating exaggeration into later phases of the optimization, t-SNE introduces more white-space between clusters which better reflects the global relations between clusters.

Standard t-SNE reveals the clustering structure at a single level of resolution. While the perplexity parameter can be used to control the trade-off between local and global structure, this can be time-consuming, and small, well-defined clusters can be missed. Alternatively, Kobak, Linderman, Steinerberger, Kluger, and Berens (2019) suggest that varying the degrees of freedom in the *t*-distribution can be used to explore the clustering structure at different levels of resolution. Modifying Eqn. (2) to $q_{ij} \propto (1 + ||\mathbf{y}_i - \mathbf{y}_j||^2/\alpha)^{-\alpha}$ allows us to use heavier-tailed distributions to model distances in the embedding space, which acts to highlight small subgoups in the resulting visualization.

3. Implementation

We introduce **openTSNE**, a comprehensive Python library that implements t-SNE and its numerous recently proposed extensions, including the implementation of efficient approximation schemes allowing the embedding of millions of data points (Van Der Maaten 2014; Linderman *et al.* 2019), the addition of new data points into existing embeddings (Poličar *et al.* 2019), various initialization improvements (Kobak and Linderman 2021), extensions (Kobak *et al.* 2019; Kobak and Berens 2019), and improved parameter defaults (Belkina *et al.* 2019). Combined, these improvements result in the faster generation of more informative visualizations.

openTSNE is compatible with the Python data science ecosystem (e.g. **numpy**, **scikit-learn**, **scanpy**), providing a familiar and intuitive API to users. Its modular design encourages extensibility and experimentation with various parameter settings and changes to the analysis pipeline. **openTSNE** aims to make the latest theoretical advances in t-SNE accessible to the wider data-science community. These features are easily accessible through an intuitive API, which closely follows the style **scikit-learn** (Buitinck, Louppe, Blondel, Pedregosa, Mueller, Grisel, Niculae, Prettenhofer, Gramfort, Grobler, Layton, VanderPlas, Joly, Holt, and Varoquaux 2013), a popular and widely used toolkit for machine learning. The following snippet exemplifies this interface and demonstrates how **openTSNE** can be used to quickly generate several visualizations, each highlighting different aspects of the underlying data structure.

```
from openTSNE import TSNE

# Vanilla t-SNE
tsne_standard = TSNE().fit(x)

# Adapt parameters to better highlight small clusters in the data
tsne_extensions = TSNE(perplexity=50, exaggeration=2, dof=0.8).fit(x)

# Use multiscale kernel for better local/global structure trade-off
from openTSNE.affinity import Multiscale

affinities = Multiscale(x, perplexities=[50, 500])
tsne_multiscale = TSNE().fit(x, affinities=affinites)

# Use the generated embeddings to add new data points
new_embedding = tsne_extensions.transform(x_new)
```

The code snippet considers a **numpy** array or **scipy** sparse matrix `x` and creates three `TSNEEmbedding` objects. These represent fully-optimized t-SNE embeddings and are internally stored as **numpy** arrays. These can be manipulated and visualized using standard Python tools. We then select one of these embeddings and use it to embed new data points from `x_new` into the embedding space, while keeping the reference embedding fixed. Additionally, **openTSNE** provides a more advanced interface that enables the user to change every aspect of the t-SNE optimization procedure. This interface is described in detail in the accompanying documentation.

4. Summary and discussion

4.1. Uncovering Structure in High-Dimensional Data

Dimensionality reduction techniques implicitly assume that high-dimensional data lies on a lower-dimensional manifold, which can accurately be captured by a small number of dimensions. However, there is no evidence that every data set can accurately be described using only two dimensions, and any such embedding will inevitably lead to a loss of information. Thus, it is beneficial to examine multiple embeddings, each of which provides a different perspective on topology and other data characteristics.

We illustrate this point by generating four different embeddings of the data on single-cell gene expression in mouse brain (Tasic *et al.* 2018). Fig. 2.a shows an embedding using default t-SNE parameters. While different clusters of excitatory and inhibitory neurons appear close to one another, all clusters appear equidistant from their neighbors, and the overall relations between groups are not obvious. The embedding in Fig. 2.b focuses on preserving larger neighborhoods of points, resulting in a more globally consistent layout where relations between clusters become more apparent. Here, it is evident from the increased white space between groups that there is one large class of excitatory neurons and two related classes of inhibitory neurons. Unfortunately, focusing on preserving large neighborhoods leads to the absorption of smaller clusters into larger ones. Alternatively, Fig. 2.c uses multi-scale similarity kernels that aim to preserve both the global organization of clusters and prevent smaller cluster absorption. We constructed the embedding from Fig. 2.d with the settings used for Fig. 2.a, but at a finer level of resolution. The figure demonstrates that some clusters are composed of numerous, smaller subgroups representing different cell subpopulations which are not visible under standard parameter settings.

When dealing with data containing millions of data points, standard t-SNE embeddings often become unwieldy – cluster boundaries are blurred, large clusters absorb smaller ones, and relationships between clusters become increasingly difficult to interpret. We constructed Fig. 3.a from the data containing expression profiles of over two million single cells captured at different time points in mouse development. The embedding indicates numerous clusters with transitions between time points, as marked by the color-coding, that are difficult to interpret. Kobak & Berens observed that increasing attractive forces between similar data points controlled via the *exaggeration* parameter leads to more compact clusters, and subsequently, more informative visualizations (Kobak and Berens 2019). For instance, 3.b doubles the default exaggeration, which uncovers some of the data's overall structure. Further doubling the exaggeration in 3.c allows us to observe that the data is comprised of two main groups of cells and eight somewhat smaller clusters. The visualization also reveals several tiny clusters, possibly corresponding to rare cell types.

Exaggeration can highlight transitions between cell states in developmental studies. Standard t-SNE often produces embeddings with clearly defined, discrete clusters. We can adjust the level of granularity and resolution of the clusters with several parameters in Fig. 2. However,

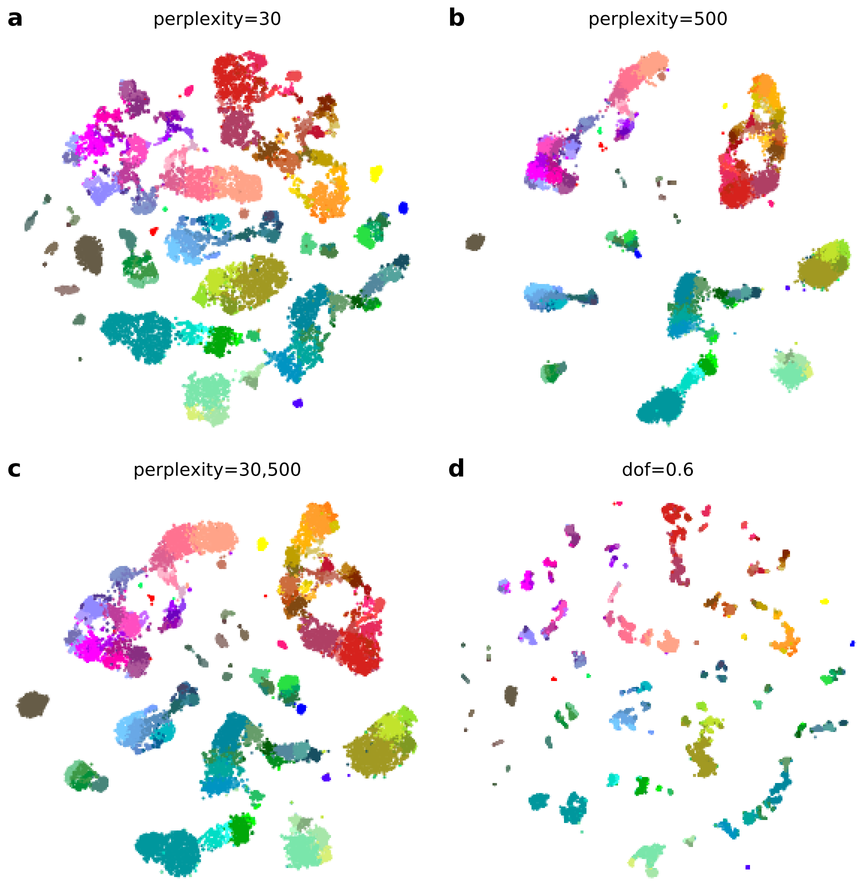

Figure 2: We use **openTSNE** to create four different visualizations of the Tasic *et al.* (2018) data, each providing a different perspective into the topology of the data. The data set contains 21,874 single-cells originating from the mouse neocortex. Cluster annotations and colors are taken from the original publication. Warm colors correspond to excitatory neurons, cool colors correspond to inhibitory neurons, and gray/brown colors correspond to non-neuronal cells. Standard t-SNE **(a)** emphasizes local structure while increasing perplexity **(b)** results in a more meaningful layout of the clusters. We can also combine the two perplexities by using a multiscale kernel affinity model **(c)** and obtain a trade-off between global and local structure. Alternatively, we can inspect more fine-grained structure and reveal smaller clusters by using a more heavy-tailed kernel **(d)**.

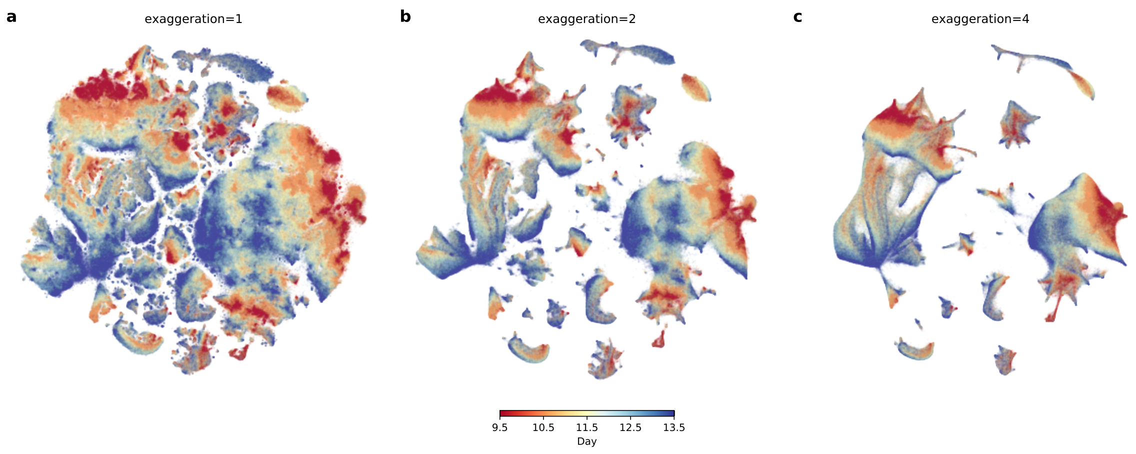

Figure 3: Increasing the exaggeration parameter leads to compact clusters, highlighting the data's global organization and emphasizing the continuous nature of cell state transitions. The data set from Cao *et al.* (2019) contains expression profiles from 2,058,652 single cells. The data were collected from mice embryos at different developmental stages at daily intervals after 9.5 to 13.5 days. **(c)** reveals that the data is comprised of two main components – the neural tube and mesenchymal cells – as well as several other smaller clusters. The colors indicate developmental progression with red indicating least-developed cells and blue indicating most developed cells. The overall developmental trajectory is most apparent with higher exaggeration levels, showing red cells slowly transitioning into blue cells. Progressively easing the exaggeration factor uncovers finer clusters within the larger groups, as shown in **(b)** with exaggeration of two and subsequently in **(a)**, where we show the standard t-SNE with no exaggeration. 32,011 putative doublets are excluded from the visualizations.

discrete clusters are often undesired in developmental studies where cells' state is assumed to follow a continuous transition path. To this end, other embedding techniques such as UMAP and ForceAtlas2 are used to better capture the continuity between cell states. Recently, Böhm *et al.* (2020) showed that embeddings produced by t-SNE with exaggeration values of 4 and ~ 30 construct embeddings which are markedly similar to UMAP and ForceAtlas2, respectively. For example, in Fig. 3.a, the developmental trajectory between different time points is difficult to observe due to many sprawled out clusters. On the other hand, it is easier to trace the development when we increase the exaggeration factor from 1 to 2 to 4 in Figs. 3.b-c.

4.2. Embedding New Samples

Unlike other popular dimensionality reduction techniques such as principal component analysis or autoencoders, t-SNE is a non-parametric method and does not define an explicit mapping to the embedding space. Therefore, embeddings of new data points need to be found through optimization (Poličar *et al.* 2019). **openTSNE** is currently the only publicly available library allowing users to add new samples to existing embeddings in a principled manner.

Figs. 1.b and 1.c demonstrate how we can use a previously labeled single-cell data set and embed cells from a separate experiment into the reference landscape. The reference data from Macosko *et al.* (2015) contains gene expression profiles from mouse retinal cells. By embedding the samples from a similar experiment on bipolar retinal cells by Shekhar *et al.* (2016), we can correctly map the bipolar cell clusters onto the reference embedding.

Embedding single cells into existing reference atlases can also be useful for cell-type classification in cases of unknown cell identities. For instance, in Fig. 4, we construct a reference embedding using labeled data from Hochgerner, Zeisel, Lönnerberg, and Linnarsson (2018) containing gene-expression profiles of cells from the mouse brain. The authors assign a type to each cell. We can verify their classification accuracy by visualizing the expression of well-established gene markers for the major cell types. We then embed cells from Harris, Hochgerner, Skene, Magno, Katona, Gonzales, Somogyi, Kessaris, Linnarsson, and Hjerling-Leffler (2018) into the constructed cell atlas. In Harris *et al.* (2018), labels are provided only for neuronal cells. In the resulting mapping, we can quickly identify other non-neuronal cell types, including oligodendrocytes and astrocytes. We can further use marker genes to validate that the mapping in the reference landscape is correct.

The examples presented above demonstrate how to use **openTSNE** to quickly gain insight into newly-sequenced, single-cell data sets by utilizing existing cell atlases. The approach is general and not limited to single-cell gene expression, and one can, in principle, apply it to any tabular data set regardless of field.

4.3. Versatility

Versatility, the ability to use and combine different optimization approaches to construct different embedding spaces, is another of **openTSNE**'s core design principles. Kobak and Berens (2019) recently provided several recommendations and tricks to obtain better and more meaningful t-SNE visualizations. These include multi-scale similarity kernels, perplexity annealing, and increasing exaggeration when working with massive data sets. **openTSNE** provides a flexible API to incorporate these improvements in just a few lines of code. Furthermore,

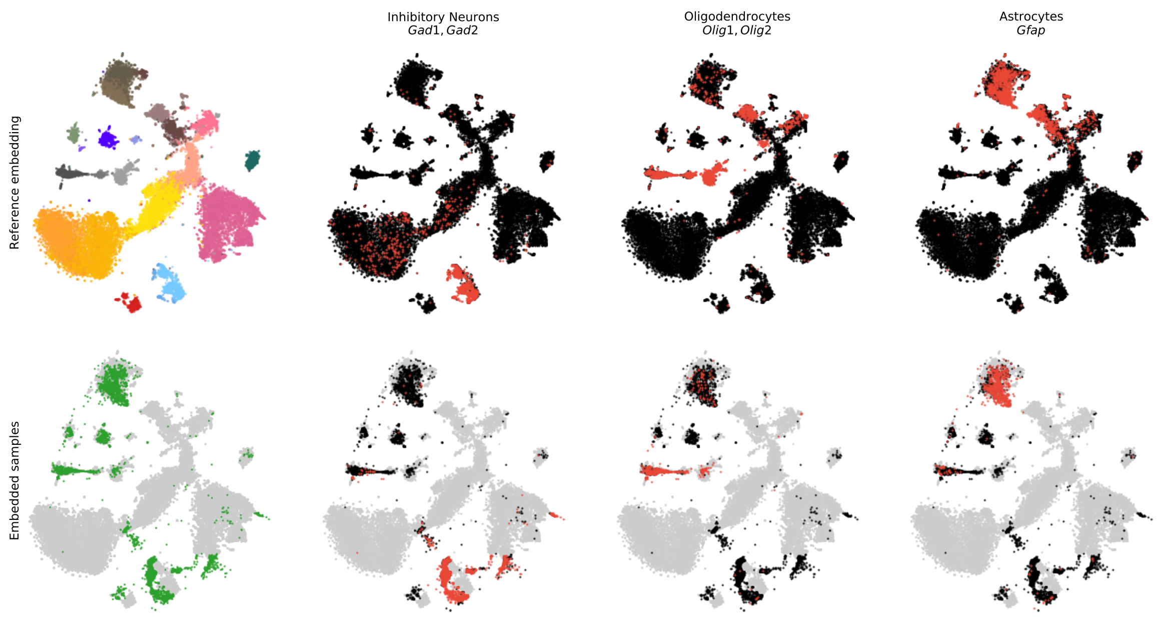

Figure 4: **openTSNE** supports embedding new samples into an existing reference t-SNE landscape. For the series of visualizations shown in this figure, we first construct a t-SNE embedding for the data from Hochgerner *et al.* (2018) containing 24,185 developing, single cells from the mouse hippocampus. The data contains gene expression in different neurons, supporting glia, and other vascular cells (upper left). Data points representing cells are colored according to cell-types assigned in the original publication; see the legend from Fig. 2 to map colors to cell-type. We then embed new, hippocampal cells collected in a study by Harris *et al.* (2018) using the embedding of Hochgerner *et al.* (2018) data as a reference. In their study, Harris *et al.* (2018) collected 6,971 single-cells and focused on identifying different types of inhibitory neurons. However, almost half of the collected cells are not neurons and were left uncharacterized. Inspecting the embeddings of these cells in the reference embedding (bottom left) reveals that in addition to inhibitory neurons, the data contains several supporting glial cells as well as a small population of endothelial cells. We can verify our approach's accuracy by inspecting marker genes for the major cell types in the reference (top row) and embedded samples (bottom row).

openTSNE supports custom affinity models, enabling users to construct t-SNE embeddings on non-tabular relational data: the only requirement imposed by the affinity-model is some notion of similarity between data points. Finally, **openTSNE**'s comprehensive callback system can be utilized to monitor and adapt different stages of the optimization phase and has been used to construct visually appealing animations of the t-SNE optimization process.

4.4. Speed

One of the t-SNE's common criticisms is limited scalability to large data sets containing, for instance, millions of data points (Becht *et al.* 2019). The culprit for slow response time stems from a particular optimization procedure and its specific implementation in popular Python libraries. Until quite recently, most popular implementations of t-SNE were based on the Barnes-Hut approximation scheme developed by Van Der Maaten (2014) with asymptotic time complexity $\mathcal{O}(N \log N)$, where N is the number of data items (*e.g.* cells). The most widely-used implementation of t-SNE came from **scikit-learn**, which exhibits long runtimes when compared to its C++ counterpart – **MulticoreTSNE**. The multi-threaded **MulticoreTSNE** implementation (Ulyanov 2016) can construct t-SNE embeddings of millions of data points in a matter of hours on widely-accessible, consumer-grade processors (Fig. 5). However, recently, Linderman *et al.* (2019) developed a new approximation scheme – FIt-SNE – which further reduces the asymptotic time complexity to $\mathcal{O}(N)$. We include this approach in **openTSNE**, enabling the embedding of large data sets in a matter of minutes.

Fig. 5 benchmarks four popular Python t-SNE implementations, including those from **scikit-learn** (v0.23.1), **MulticoreTSNE** (v0.1), **FIt-SNE** (v1.1.0), and our **openTSNE** (v0.6.0). We perform benchmarks on two computational platforms, one representing the usage on a personal computer and the other the utility of these libraries on a high-performance computing platform. The Intel(R) Core i7 is commonly found in consumer-grade laptop computers, while Intel(R) Xeon(R) processors appear in high-performance computing machines. Benchmarks were run for $1,000$ iterations with the original t-SNE parameters, as some implementations do not allow their modification.

Benchmark results (Fig. 5) confirm that both **FIt-SNE** and **openTSNE** scale better than their Barnes-Hut counterparts – **scikit-learn** and **MulticoreTSNE**. While **openTSNE**'s implementation uses Python which incurs some runtime overhead compared to its C++ counterpart **FIt-SNE**, the two libraries' speeds are surprisingly comparable. Modern computer processors contain multiple cores, allowing us to use multi-threading, which further reduces the gap between **openTSNE** and **FIt-SNE**. On the server-grade processor, **openTSNE** is slightly faster than its pure C++ counterpart when utilizing multiple cores. **openTSNE** uses **numpy** for most linear algebra operations, which may make better use of the Intel Math Kernel Library (MKL), which is aggressively optimized on Intel(R) Xeon(R) processors.

openTSNE provides a flexible API, allowing splitting up the embedding-construction process into several parts, caching slow operations. Running t-SNE optimization in stages enables users to quickly experiment with different parameter settings and iterate on their final visualizations.

4.5. Ease of Use

Intuitive access and simple installation procedures almost universally correlate with the widespread adoption of novel computational techniques. While the t-SNE implementation

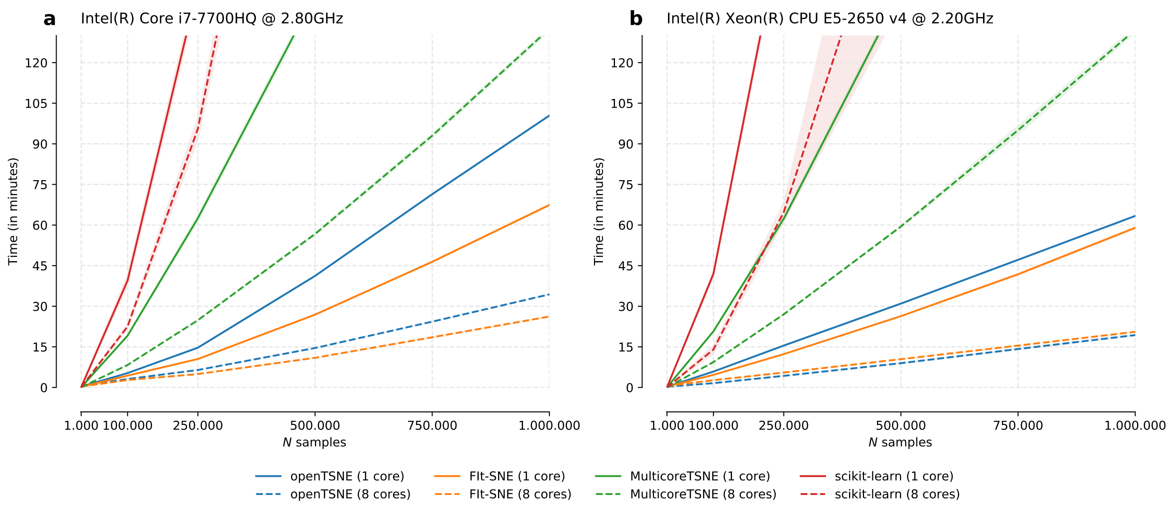

Figure 5: We benchmark **openTSNE** (v0.6.0) against three popular open-source implementations from **scikit-learn** (Pedregosa *et al.* 2011) (v0.23.1), **MulticoreTSNE** (Ulyanov 2016) (v0.1), and **FIt-SNE** (Linderman *et al.* 2019) (v1.1.0). Experiments were run on a consumer-grade Intel Core i7-7700HQ processor found in laptop computers, and on a server-grade Intel Xeon E5-2650. To generate benchmark data sets of different sizes, we subsampled data from the 10X Genomics 1.3 million mouse brain data set five times, resulting in five different data sets for each size. In total, we run each implementation on 30 different data sets. Notice that **openTSNE** scales similarly to **FIt-SNE**, as they both use the same interpolation-based approximation scheme, while **scikit-learn** and **MulticoreTSNE** utilize the Barnes-Hut approximation.

from **scikit-learn** fits this requirement, the implementation proves prohibitively slow for even moderately-sized data sets that span tens of thousands of data records. Other C++ implementations such as **MulticoreTSNE** and **FIt-SNE** exhibit better scaling in more massive data sets, but do not provide precompiled binaries and require users to compile the software themselves. This problem is critical, for instance, for users of the Windows operating system, where the C++ compiler does not come with the system, making the correct configuration of current t-SNE implementations cumbersome.

We designed **openTSNE** to be accessible to a broader audience. We provide precompiled binaries for all major Python versions on all major platforms, making the installation process as seamless as possible. One can install **openTSNE** through the Python Package Index (**PyPI**) or **conda** from the **conda-forge** channel, the two most widely adopted Python package managers. **openTSNE**'s interface is inspired by **scikit-learn**, which is well established in the Python data science ecosystem. **openTSNE** implements multi-threaded versions of both the Barnes-Hut and FIt-SNE approximation schemes, enabling it to be applied to data sets containing millions of data points. While the Python virtual machine inevitably introduces some performance overhead, the runtime is comparable to its C++ counterpart – **FIt-SNE**. Finally, **openTSNE** is extensible. Its modular design enables researchers to quickly experiment with different parameter settings and easily incorporate custom components into the software. We provide a full feature list and comparison to other popular t-SNE implementations in Table 1.

	scikit-learn	**BHTSNE**	**MulticoreTSNE**	**FIt-SNE**	**openTSNE**
Ease of Installation					
Native Python installation	✓				✓
PyPI package	✓		✓		✓
conda package	✓				✓
Approximation Schemes					
Barnes-Hut ($\mathcal{O}(N \log N)$)	✓	✓	✓		✓
FIt-SNE ($\mathcal{O}(N)$)				✓	✓
Advanced Features and Extensions					
Multiscale Gaussian kernels				✓	✓
Fully-custom affinity kernels					✓
Variable degrees of freedom				✓	✓
Variable exaggeration				✓	✓
Globally-consistent initialization				✓	✓
Automatic learning rate				✓	✓
Adding samples to embeddings					✓
Callback system					✓
Interactive optimization					✓
Project Quality					
User documentation	✓				✓
End-to-end usage examples	✓			✓	✓
API reference	✓				✓
Continuous integration	✓		✓		✓

Table 1: We compare the features of **openTSNE** (v0.6.0) to four popular open-source t-SNE implementations from **scikit-learn** (v0.23.1), **BHTSNE** (master), **MulticoreTSNE** (v0.1), and **FIt-SNE** (v1.1.0).

5. Conclusion

We introduce **openTSNE**, the most feature-complete, open-source, Python implementation of the widely popular t-SNE visualization algorithm. The library incorporates the latest developments to the t-SNE algorithm and makes them easily accessible through a familiar API. **openTSNE** implements efficient approximation schemes, making it suitable for the visualization of large data sets, containing up to millions of data points. Finally, **openTSNE** is easy to install and comes with precompiled binaries available for all major platforms.

Availability and Documentation

openTSNE is distributed under the BSD-3-Clause License and its source code is publicly available at `https://github.com/pavlin-policar/openTSNE`. **openTSNE** can also be installed from **PyPI** and **conda-forge**. End-to-end usage examples, documentation, and API reference are all available at `https://opentsne.readthedocs.io`.

Computational details

The results in this paper were obtained using Python 3.7.6 with the **openTSNE** XXX package. **openTSNE** has three direct dependencies: **numpy**, **scipy**, and **scikit-learn**. In our experiments, we used **numpy** v1.18.1, **scipy** v1.4.1. **scikit-learn** v0.23.1. As these packages include their own downstream dependencies, the full environment used to produce the figures is included in the accompanying GitHub repository `https://github.com/pavlin-policar/opentsne-paper`.

Acknowledgments

We want to thank Dmitry Kobak for helpful discussions regarding t-SNE and his contributions to the source code. We would also like to thank George Linderman for his help with the FIt-SNE algorithm. This work was supported by Slovenian Research Agency (P2-0209).

References

Beaulaurier J, Zhu S, Deikus G, Mogno I, Zhang XS, Davis-Richardson A, Canepa R, Triplett EW, Faith JJ, Sebra R, *et al.* (2018). “Metagenomic binning and association of plasmids with bacterial host genomes using DNA methylation.” *Nature Biotechnology*, **36**(1), 61.

Becht E, McInnes L, Healy J, Dutertre CA, Kwok IW, Ng LG, Ginhoux F, Newell EW (2019). “Dimensionality reduction for visualizing single-cell data using UMAP.” *Nature Biotechnology*, **37**(1), 38.

Belkina AC, Ciccolella CO, Anno R, Halpert R, Spidlen J, Snyder-Cappione JE (2019). “Automated optimized parameters for T-distributed stochastic neighbor embedding improve visualization and analysis of large datasets.” *Nature Communications*, **10**(1), 1–12.

Böhm JN, Berens P, Kobak D (2020). “A Unifying Perspective on Neighbor Embeddings along the Attraction-Repulsion Spectrum.” *arXiv preprint arXiv:2007.08902*.

Buitinck L, Louppe G, Blondel M, Pedregosa F, Mueller A, Grisel O, Niculae V, Prettenhofer P, Gramfort A, Grobler J, Layton R, VanderPlas J, Joly A, Holt B, Varoquaux G (2013). "API design for machine learning software: experiences from the scikit-learn project." In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122.

Cao J, Spielmann M, Qiu X, Huang X, Ibrahim DM, Hill AJ, Zhang F, Mundlos S, Christiansen L, Steemers FJ, *et al.* (2019). "The single-cell transcriptional landscape of mammalian organogenesis." *Nature*, **566**(7745), 496–502.

Ding J, Condon A, Shah SP (2018). "Interpretable dimensionality reduction of single cell transcriptome data with deep generative models." *Nature Communications*, **9**(1), 2002.

Harris KD, Hochgerner H, Skene NG, Magno L, Katona L, Gonzales CB, Somogyi P, Kessaris N, Linnarsson S, Hjerling-Leffler J (2018). "Classes and continua of hippocampal CA1 inhibitory neurons revealed by single-cell transcriptomics." *PLoS Biology*, **16**(6), e2006387.

Hirata J, Hosomichi K, Sakaue S, Kanai M, Nakaoka H, Ishigaki K, Suzuki K, Akiyama M, Kishikawa T, Ogawa K, *et al.* (2019). "Genetic and phenotypic landscape of the major histocompatibilty complex region in the Japanese population." *Nature Genetics*, **51**(3), 470–480.

Hochgerner H, Zeisel A, Lönnerberg P, Linnarsson S (2018). "Conserved properties of dentate gyrus neurogenesis across postnatal development revealed by single-cell RNA sequencing." *Nature Neuroscience*, **21**(2), 290–299.

Jacobs RA (1988). "Increased rates of convergence through learning rate adaptation." *Neural Networks*, **1**(4), 295–307.

Jacomy M, Venturini T, Heymann S, Bastian M (2014). "ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software." *PloS One*, **9**(6), e98679.

Kobak D, Berens P (2019). "The art of using t-SNE for single-cell transcriptomics." *Nature Communications*, **10**(1), 1–14.

Kobak D, Linderman G (2021). "Initialization is critical for preserving global data structure in both t-SNE and UMAP." *Nature Biotechnology*, **39**(2), 156–157.

Kobak D, Linderman G, Steinerberger S, Kluger Y, Berens P (2019). "Heavy-tailed kernels reveal a finer cluster structure in t-SNE visualisations." In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 124–139. Springer.

Linderman G, Rachh M, Hoskins JG, Steinerberger S, Kluger Y (2019). "Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data." *Nature Methods*, **16**(3), 243–245.

Macosko EZ, Basu A, Satija R, Nemesh J, Shekhar K, Goldman M, Tirosh I, Bialas AR, Kamitaki N, Martersteck EM, *et al.* (2015). "Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets." *Cell*, **161**(5), 1202–1214.

McInnes L, Healy J, Melville J (2018). "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction." *ArXiv e-prints*. 1802.03426.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, *et al.* (2011). "Scikit-learn: Machine learning in Python." *The Journal of Machine Learning Research*, **12**, 2825–2830.

Poličar PG, Stražar M, Zupan B (2019). "Embedding to reference t-SNE space addresses batch effects in single-cell classification." In *International Conference on Discovery Science*, pp. 246–260. Springer.

Shekhar K, Lapan SW, Whitney IE, Tran NM, Macosko EZ, Kowalczyk M, Adiconis X, Levin JZ, Nemesh J, Goldman M, *et al.* (2016). "Comprehensive classification of retinal bipolar neurons by single-cell transcriptomics." *Cell*, **166**(5), 1308–1323.

Sheth RU, Li M, Jiang W, Sims PA, Leong KW, Wang HH (2019). "Spatial metagenomic characterization of microbial biogeography in the gut." *Nature Biotechnology*, **37**(8), 877–883.

Tasic B, Yao Z, Graybuck LT, Smith KA, Nguyen TN, Bertagnolli D, Goldy J, Garren E, Economo MN, Viswanathan S, *et al.* (2018). "Shared and distinct transcriptomic cell types across neocortical areas." *Nature*, **563**(7729), 72–78.

Tkachev A, Stepanova V, Zhang L, Khrameeva E, Zubkov D, Giavalisco P, Khaitovich P (2019). "Differences in lipidome and metabolome organization of prefrontal cortex among human populations." *Scientific Reports*, **9**(1), 1–10.

Ulyanov D (2016). "Multicore-TSNE." `https://github.com/DmitryUlyanov/Multicore-TSNE`.

Van Der Maaten L (2014). "Accelerating t-SNE using tree-based algorithms." *The Journal of Machine Learning Research*, **15**(1), 3221–3245.

Van Der Maaten L, Hinton G (2008). "Visualizing data using t-SNE." *Journal of Machine Learning Research*, **9**(Nov), 2579–2605.

Affiliation:

Pavlin G. Poličar
Faculty of Computer and Information Science, University of Ljubljana
Večna pot 113, Ljubljana, Slovenia
E-mail: `pavlin.policar@fri.uni-lj.si`

Journal of Statistical Software `http://www.jstatsoft.org/`
published by the Foundation for Open Access Statistics `http://www.foastat.org/`
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
`doi:10.18637/jss.v000.i00` *Accepted:* yyyy-mm-dd