SOFTWARE

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

Pavlin G. Poličar[1*], Martin Stražar[1,2] and Blaž Zupan[1,3]

Abstract

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.

Keywords: sample; article; author

Background

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 [1]. Among these, t-distributed stochastic neighbor embedding (t-SNE) [2] lately received much attention as it can address high volumes of data and reveal meaningful clustering structure. Most of the recent reports on single-cell gene expression data start with an overview of the cell landscape, where t-SNE embeds high-dimensional expression profiles into a two-dimensional space [3, 4, 5]. Fig. **??**.a presents an example of one such embedding.

Despite its utility, t-SNE has been criticized for poor scalability when addressing large data sets, lack of global organization *t-SNE focuses on local clusters that are arbitrarily scattered in the low-dimensional space* and absence of theoretically-founded implementations to map new data into existing embeddings [**?**, **?**]. Most of these shortcomings, have recently been addressed. [**?**] sped-up the method through an interpolation-based approximation, reducing the time complexity to be merely linear dependent on the number of samples. [**?**] 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, [**?**] proposed a related approach using neural networks.

Results

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.

The following code is used to generate Fig. 1

```
# Perplexity 30 (standard t-SNE)
perp_30 = openTSNE.TSNE(
    perplexity=30, metric="cosine"
).fit(X)

# Perplexity 500 (emphasise global structure)
perp_500 = openTSNE.TSNE(
    perplexity=500, metric="cosine"
).fit(X)

# Multiscale kernel with perplexities 30,500
affinities = openTSNE.affinity.Multiscale(
    X, perplexities=[30, 500], metric="cosine"
)
init = openTSNE.initialization.pca(X)
ms = openTSNE.TSNEEmbedding(init, affinities)
ms = ms.optimize(n_iter=250, exaggeration=12)
ms = ms.optimize(n_iter=250, exaggeration=1)

# Expose finer structure using smaller dof
perp_30 = openTSNE.TSNE(
    perplexity=30, metric="cosine", dof=0.6
).fit(X)
```

*Correspondence: pavlin.policar@fri.uni-lj.si
[1]Faculty of Computer and Information Science, University of Ljubljana, SI 1000 Ljubljana, Slovenia
Full list of author information is available at the end of the article

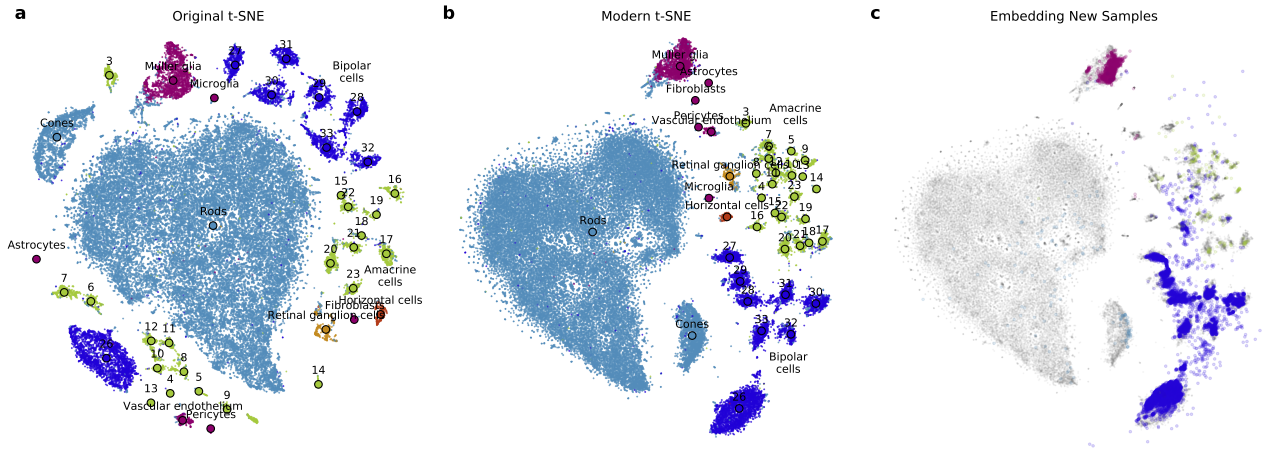

Figure 1

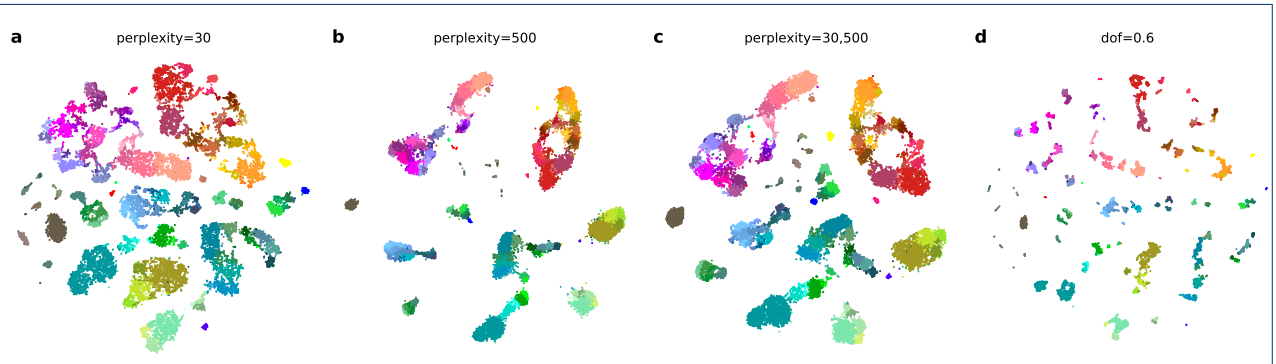

Figure 2 We use openTSNE to create four different visualizations of the Tasic *et al.* [5] data set containing 21,874 single-cells originating from the mouse neocortex. Cluster annotations and colors are taken from the original publication. Warm colors correspond to excitatory neurons, cool colors correspond to inhibitory neurons, and gray/brown colors correspond to non-neuronal cells. The 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 good 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)**.

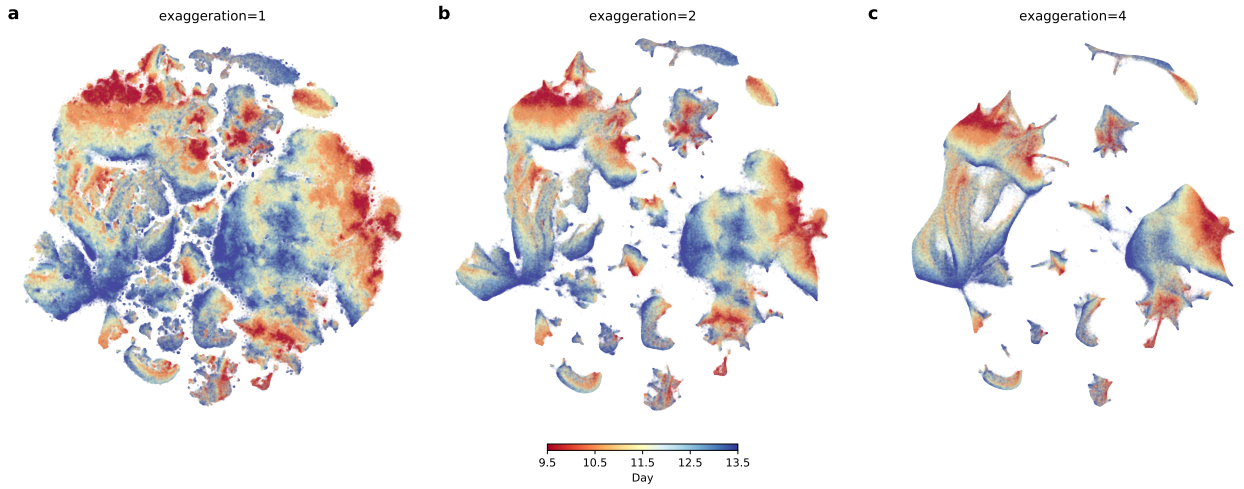

Figure 3 The data set from Cao *et al.* [4] contains expression profiles from 2,058,652 single cells. The data were collected from mice embryos at different developmental time-points at daily intervals after 9.5 to 13.5 days. The exaggeration parameter can be used to highlight the global organization of the clusters and enforce cluster compactness. **(a)** shows the data is comprised of two main components – the neural tube and mesenchymal cells – as well as several other smaller clusters. Additionally, 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 levels of exaggeration, 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 2 and subsequently in **(c)**, where we show the standard t-SNE with no exaggeration. 32,011 putative doublets are excluded from the visualizations.

The following code is used to generate Fig. 2

```
indices = np.random.permutation(list(range(x.shape[0])))
reverse = np.argsort(indices)
# Split the data into a sample/remainder sets
x_sample, x_rest = x[indices[:25000]], x[indices[25000:]]
y_sample, y_rest = y[indices[:25000]], y[indices[25000:]]
# Create an initial embedding using a subset of the data
sample_embedding = openTSNE.TSNE(
    initialization="spectral",
    perplexity=500,
    metric="cosine",
).fit(x_sample)
# Position the remainder of the points to their nearest
# neighbor in the sample embedding
rest_init = sample_embedding.prepare_partial(x_rest, k=1)
init_full = np.vstack(
    [sample_embedding, rest_init]
)[reverse]
init_full = openTSNE.initialization.rescale(init_full)
# Calculate the affinity model using all the data points
affinities = openTSNE.affinity.PerplexityBasedNN(
    x, perplexity=30, metric="cosine"
)

embedding = openTSNE.TSNEEmbedding(
    init_full, affinities
%)
exag12 = embedding.optimize(n_iter=500, exaggeration=12)
exag4 = exag12.optimize(n_iter=500, exaggeration=4)
exag2 = exag4.optimize(n_iter=500, exaggeration=2)
exag1 = exag2.optimize(n_iter=500, exaggeration=1)
```

The following code is used to generate Fig. 3

```
# Assuming we have prepared a reference embedding and
# assuming we have aligned new_X to the reference X matrix
new_embedding = reference_embedding.transform(new_X)
```

We want to mention that this works for mitigating batch effects in our earlier work [8].

Discussion

Text for this section ...

Conclusion

Text for this section ...

Methods

t-SNE

Local, non-linear dimensionality reduction by t-SNE is performed as follows. 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 reference 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 multi-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 t-SNE is defined as:

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

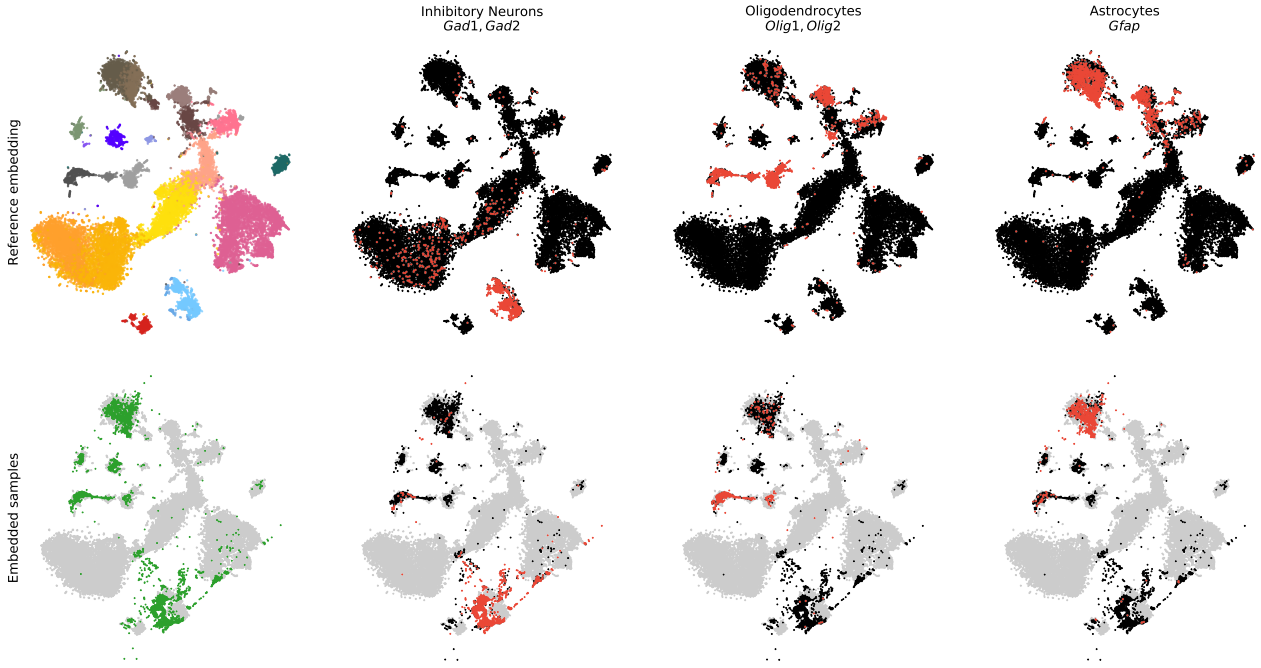

Figure 4 openTSNE enables the user to embed new samples into an existing, reference embedding. We construct a t-SNE embedding for the data from Hochgerner *et al.* [6] containing 24,185 developing, single cells from the mouse hippocampus. The data contains different types of neurons, supporting glia, and other vascular cells (upper left). Cell annotations are taken from the original publications. The colors correspond to Fig. 1. We then embed new, hippocampal cells collected by Harris *et al.* [7] using the previous embedding as a reference. In their study, Harris *et al.* 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 where these cells are embedded in the reference embedding (bottom left) reveals that in addition to inhibitory neurons, the data contains a number of supporting glia as well as a small population of endothelial cells. We can verify the accuracy of our approach by inspecting marker genes for the major cell types in the reference (top row) and embedded samples (bottom row).

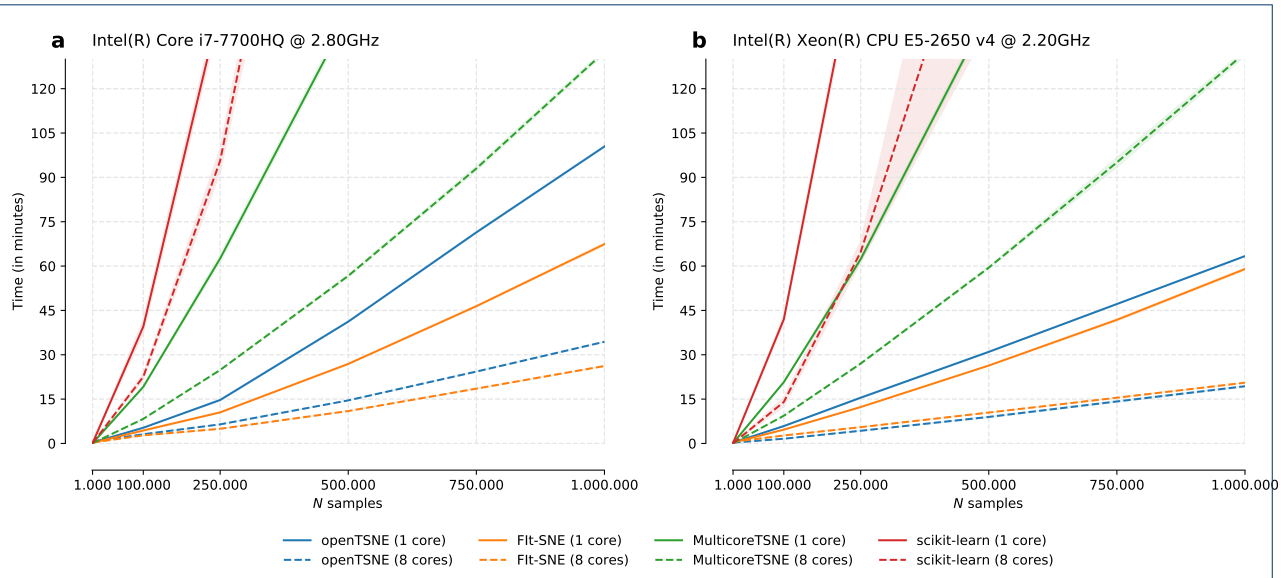

Figure 5 We benchmark openTSNE against three popular open-source implementations including scikit-learn [9], MulticoreTSNE [10], and FIt-SNE [11]. Benchmarks were run on a consumer-grade Intel Core i7-7700HQ processor found in laptop computers, and on a server-grade Intel Xeon E5-2650. To generate benchmark 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, each implementation was run 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.

where $\mathcal{D}$ is a distance measure. This is then symmetrized to

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

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

$$\text{Perplexity} = 2^{H(P_i)} \tag{3}$$

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

$$H(P_i) = -\sum_i p_{j|i} \log_2(p_{j|i}). \tag{4}$$

Different bandwidths σ_i enable t-SNE to adapt to the varying density of the data in the multi-dimensional space.

The similarity between points $\mathbf{y}_i$ and $\mathbf{y}_j$ in the embedding space is defined using the t-distribution with one degree of freedom

$$q_{ij} = \frac{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1}}{\sum_{k \neq l} \left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)^{-1}}, \quad q_{ii} = 0. \tag{5}$$

The t-SNE method finds an embedding $\mathbf{Y}$ that minimizes the Kullback-Leibler (KL) divergence between $\mathbf{P}$ and $\mathbf{Q}$,

$$C = \text{KL}(\mathbf{P} \mid\mid \mathbf{Q}) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{6}$$

It is then easy to show that the corresponding gradient takes on the form

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} \left(p_{ij} - q_{ij}\right) \left(\mathbf{y}_i - \mathbf{y}_j\right) \left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1}. \tag{7}$$

Optimization is performed using batch gradient descent with the delta-bar-delta update rule for increased convergence speed [12]. Originally, t-SNE was run for 1000 iterations consisting of two phases. In the first

early exaggeration phase, the attractive forces between data points is increased by a factor $\rho = 12$ so that points can move throughout the embedding uninhibited by repulsive forces. Then the remaining 750 iterations are run with $\rho = 1$, which reverts the attractive forces to their original values.

Belkina *et al.* later found that the number of iterations can saftely be decreased to 750 iterations by increasing the learning rate to $\eta = N/12$ instead of the default and somewhat arbitrarily set $\eta = 200$ [13]. A welcome side effect of this reduction is also faster runtime.

Efficient Implementation

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 beckons for the development of efficient approximation schemes. Van der Maaten observed that the t-SNE gradient may be reformulated in terms of a N-body problem where data points may be interpreted as particles which attract and repel each other [14]. The gradient may be rewritten as

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \left[\sum_{j \neq i} p_{ij} q_{ij} Z \left(\mathbf{y}_i - \mathbf{y}_j \right) - \sum_{j \neq i} q_{ij}^2 Z \left(\mathbf{y}_i - \mathbf{y}_j \right) \right], \tag{8}$$

where $Z = \sum_{k \neq l} \left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)^{-1}$. The two terms may be viewed as the attractive and repulsive forces between particles, respectively. Each term lends itself to efficient approximations, which enable us to greatly reduce the time complexity of t-SNE.

Attractive Forces

Van der Maaten observed that since p_{ij} is calculated in the high-dimensional space using a Gaussian kernel, and that the bandwidth is selected such that only a fixed number of closest neighbors (determined by the perplexity parameter) will contribute to the attractive forces of a given point due to the exponential decay of the Gaussian kernel. Therefore it is sufficient to calculate the attractive forces only over a relatively small number of nearest neighbors instead of all N points. By utilizing tree-based nearest-neighbor search methods, the time complexity is thus reduced to $\mathcal{O}(N \log N)$. Linderman *et al.* further realised that, qualitatively, embeddings are visually virtually indistinguishable when using only *approximate* nearest neighbors, further reducing time complexity to $\mathcal{O}(N)$ [11].

Repulsive Forces

Examining the second term of Eqn. (8) we notice that each point indiscriminately exerts a repulsive force on all other points. Van der Maaten proposed an approach based on N-body simulations and used a space-partitioning Barnes-Hut tree approach to approximate the interaction between data points. Briefly, the approach splits the space into quadrants (in the 2D case) and if a given point is far enough away from a quadrant, the points in that quadrant are summarized by a single center of mass. This reduces the time complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.

More recently, Linderman *et al.* proposed an alternative approach, FIt-SNE, based on non-uniform convolutions to calculating all pairwise interactions between repelling data points. Briefly, Linderman *et al.* observed that the repulsive forces $\mathbf{R}$ may be rewritten as

$$\begin{aligned} \mathbf{R}_i &= \sum_{j \neq i} q_{ij}^2 Z \left(\mathbf{y}_i - \mathbf{y}_j \right) \\ &= \sum_{j \neq i} \frac{\mathbf{y}_i - \mathbf{y}_j}{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^2} \Bigg/ \sum_{k \neq l} \frac{1}{1 + ||\mathbf{y}_k - \mathbf{y}_l||^2} \end{aligned} \tag{9}$$

and computed by evaluating three terms

$$\begin{aligned} \phi_{1,j} &= \sum_{j \neq i} \frac{1}{1 + ||\mathbf{y}_j - \mathbf{y}_i||^2}, \\ \phi_{2,j} &= \sum_{j \neq i} \frac{\mathbf{y}_j}{\left(1 + ||\mathbf{y}_j - \mathbf{y}_i||^2\right)^2}, \\ \phi_{3,j} &= \sum_{j \neq i} \frac{1}{\left(1 + ||\mathbf{y}_j - \mathbf{y}_i||^2\right)^2}. \end{aligned} \tag{10}$$

Then the numerator and denominator of Eqn. (9) can be computed as $\mathbf{y}_i \phi_{1,j} - \phi_{2,j}$ and $Z = \sum_j \phi_{3,j}$, respectively. These interactions are accelerated by interpolating these terms through a grid of equispaced interpolation points. This shifts the computational burden onto the interpolation points and reduces the time complexity to $\mathcal{O}(N)$.

openTSNE provides efficient implementations for all the aforementioned approximations but defaults to using approximate nearest neighbor search and the non-uniform convolutions approach as these enable the user to quickly create informative visualizations for data sets containing up to millions of samples. We would note here that while the default choices introduce some runtime overhead on smaller data sets, this effect is minimal and impacts runtime by only a few seconds compared to the exact nearest-neighbor search Barnes-Hut approximation.

Embedding New Samples

Unlike other popular dimensionality reduction methods 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 the use of optimization techniques [8]. When adding new data points to an existing, reference embedding, the reference data points are fixed in place and the 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}

$$p_{j|i} = \frac{\exp\left(-\frac{1}{2}\mathcal{D}(\mathbf{x}_i, \mathbf{v}_j)/\sigma_i^2\right)}{\sum_i \exp\left(-\frac{1}{2}\mathcal{D}(\mathbf{x}_i, \mathbf{v}_j)/\sigma_i^2\right)}, \tag{11}$$

$$q_{j|i} = \frac{\left(1 + ||\mathbf{y}_i - \mathbf{w}_j||^2\right)^{-1}}{\sum_i \left(1 + ||\mathbf{y}_i - \mathbf{w}_j||^2\right)^{-1}}, \tag{12}$$

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 secondary 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. (2). The gradients of $\mathbf{w}_j$ with respect to the loss (Eqn. 6) are:

$$\frac{\partial C}{\partial \mathbf{w}_j} = 2\sum_i \left(p_{j|i} - q_{j|i}\right)\left(\mathbf{y}_i - \mathbf{w}_j\right)\left(1 + ||\mathbf{y}_i - \mathbf{w}_j||^2\right)^{-1} \tag{13}$$

As in 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 $\mathcal{O}(\max\{N, M\})$, respectively. In the FIt-SNE approximation scheme, we additionally exploit the fact that the reference embedding remains fixed throughout the optimization of newly added points, and precompute the interpolation grid. This further reduces the runtime complexity from $\mathcal{O}(\max\{N, M\})$ to $\mathcal{O}(M)$.

openTSNE is currently the only open-source t-SNE implementation allowing users to add new samples into existing embeddings.

Alternative Perplexity Kernels

In standard t-SNE, distances are converted to similarities through the use of Gaussian kernels of varying bandwidths. The bandwidths are indirectly determined by the user-provided perplexity parameter, so that a fixed number of nearest data points will have non-zero similarity. One common trick for uncovering the global relations between clusters is to increase perplexity such that more long-range interactions are preserved in the final embedding. However, one unfortunate side effect of increasing perplexity is that smaller clusters get absorbed into larger ones, and less overall clusters are revealed.

Kobak *et al.* [15] suggest that replacing the Gaussian kernel with a mixture of Gaussians may provide better insight into both the local and global structure. Therefore, the affinities in the input space become

$$p_{j|i} \propto \frac{1}{\sigma_{1,i}} \exp\left(-\mathcal{D}(\mathbf{x}_i, \mathbf{x}_j)/2\sigma_{1,i}^2\right) + \frac{1}{\sigma_{2,i}} \exp\left(-\mathcal{D}(\mathbf{x}_i, \mathbf{x}_j)/2\sigma_{2,i}^2\right). \tag{14}$$

The bandwidths of each Gaussian $\sigma_{1,i}$ and $\sigma_{2,i}$ is determined by selecting different perplexity values. They show that using a kernel with perplexity 500 captures the long-range interactions which preserve global cluster organization in the final embedding, and combine this with a kernel with perplexity 50, which prevents small, well-defined clusters from being absorbed into larger ones.

Variable Degrees of Freedom

Standard t-SNE is often used to reveal clustering structure in data at one level of resolution. Different perplexity parameter values can be used to identify global cluster relationships or small, well-isolated groups. Unfortunately, varying perplexity values can be time-consuming as this involves recomputing the KNNG which is often the most expensive part of the t-SNE algorithm. Alternatively, Kobak *et al.* [16] suggest that varying the degree of freedom in the t-distribution used in the embedding space during optimization can be used to explore the clustering structure at different levels of resolution.

Standard t-SNE models similarities between data points in the embedding space using a t-distribution with a single degree of freedom

$$q_{ij} \propto \left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2/\alpha\right)^{-\alpha} = \frac{1}{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2/\alpha\right)^{\alpha}}. \tag{15}$$

In standard t-SNE $\alpha = 1$ so this simplifies to Eqn. (5).

The gradient of the loss function then becomes

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4\sum_{j \neq i} \left(p_{ij} - q_{ij}\right) q_{ij}^{1/\alpha} \left(\mathbf{y}_i - \mathbf{y}_j\right), \tag{16}$$

which can, again, be decomposed into the attractive and repulsive forces as

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} p_{ij} q_{ij}^{1/\alpha} (\mathbf{y}_i - \mathbf{y}_j) - 4 \sum_{j \neq i} q_{ij}^{\frac{\alpha+1}{\alpha}} / Z (\mathbf{y}_i - \mathbf{y}_j). \quad (17)$$

Adapting existing approximation schemes to this formulation is straightforward. We provide efficient implementations of both the Barnes-Hut and the FIt-SNE approximation schemes, where we replace the terms from Eqn. (10) with

$$\phi_{1,j} = \sum_{j \neq i} \frac{1}{(1 + ||\mathbf{y}_j - \mathbf{y}_i||^2/\alpha)^{\alpha+1}},$$
$$\phi_{2,j} = \sum_{j \neq i} \frac{\mathbf{y}_j}{(1 + ||\mathbf{y}_j - \mathbf{y}_i||^2/\alpha)^{\alpha+1}},$$
$$\phi_{3,j} = \sum_{j \neq i} \frac{1}{(1 + ||\mathbf{y}_j - \mathbf{y}_i||^2/\alpha)}.$$

Variable Exaggeration

Embeddings produced by standard t-SNE often produce clusters separated by thin boundaries and make use of all available space. While this is often desirable, this often obscures the global relationships between clusters as all neighboring clusters appear at the same distance from one another. Other dimensionality reduction methods such as UMAP [1] or ForceAtlas2 [17] tend to produce embeddings where clusters appear more compact and the white-space separating the clusters may be interpreted at least partially as a loose measure of distance.

Kobak *et al.* [**?**] showed that the early exaggeration factor ρ in t-SNE may be increased to produce more compact layouts often seen in UMAP or ForceAtlas2. Early exaggeration is a trick introduced by van der Maaten [2] used in the first 250 iterations and increases the attractive forces between data points, enabling neighboring points to move close to one another while ignoring the repulsive forces between data points. In standard t-SNE, the remainder of the iterations are run with $\rho = 1$. Kobak *et al.* showed that using $\rho = 4$ and $\rho = 30$ in the second phase of the optimization result in embeddings visually similar to UMAP embeddings, and ForceAtlas2 embeddings, respectively.

While it is difficult to claim one is better than the other, different settings of ρ may uncover different properties and clustering structures. For example, in single-cell data, standard t-SNE with $\rho = 1$ often uncovers distinct groups of cell-types but obscures developmental paths. This problem is exacerbated when dealing with large numbers of data points. Conversely, ForceAtlas2 has often been used to uncover trajectories and transitions between cell types, but larger clusters often absorb small, distinct groups of cells.

Alternative Initialization Schemes

Ease of Use

Widespread adoption of novel techniques is almost universally accompanied by easy-to-use and easy-to-install software. While the t-SNE implementation in scikit-learn and Rtsne fits this requirement, it implements the outdated Barnes-Hut approximation scheme which is unsuitable for larger data sets containing up to millions of data points. Other C++ implementations such as MulticoreTSNE and FIt-SNE are more suitable for larger data sets, but do not provide precompiled binaries to their software, requiring users to compile the software themselves. This is problematic from a usability standpoint, as many users often either do not know how to compile C++ source code or use Windows, which does not include a C++ compiler by default.

openTSNE is a Python package which focuses on ease-of-use and extensibility. We provide precompiled binaries for all major Python version on all major platforms, making the installation process as seamless as possible. openTSNE can be installed through the Python Package Index (PyPI) or through conda on the conda-forge channel. openTSNE is designed to be familliar to Python users and loosely follows the scikit-learn API, which is well established in the Python data-science ecosystem. openTSNE implements both the Barnes-Hut and FIt-SNE approximation schemes, allowing it to be used on data sets containing millions of data points. While the Python virtual machine introduces some overhead, the runtime is comparable to that of its pure C++ counterpart – FIt-SNE. Finally, openTSNE is extensible. Its modular interface enables researchers to quickly experiment with different settings and easily incorporate their own components into the software.

Alternative initialization schemes Spectral and PCA, cite dmitry and george paper

Incorporates latest developments $lr = N/12, n_iter = 750$

Availability

Text for this section ...

Competing interests

The authors declare that they have no competing interests.

Author's contributions

Text for this section ...

	scikit-learn	MulticoreTSNE	Fit-SNE	openTSNE
PyPI package	✓	✓		✓
conda package	✓			✓
Precompiled binaries	✓			✓
$\mathcal{O}(N \log N)$	✓	✓		✓
$\mathcal{O}(N)$			✓	✓
Variable degrees of freedom			✓	✓
Variable exaggeration			✓	✓
Multiscale Gaussian kernels			✓	✓
Custom affinity kernels				✓
Adding new samples				✓

Acknowledgements
Text for this section . . .

Author details
[1]Faculty of Computer and Information Science, University of Ljubljana, SI 1000 Ljubljana, Slovenia. [2]Broad Institute of Harvard and MIT, MA 02142 Cambridge, U.S.A.. [3]Department of Molecular and Human Genetics, Baylor College of Medicine, TX 77030 Houston, U.S.A..

References

1. McInnes, L., Healy, J., Melville, J.: UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. ArXiv e-prints (2018). 1802.03426
2. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of Machine Learning Research **9**(Nov), 2579–2605 (2008)
3. Macosko, E.Z., Basu, A., Satija, R., Nemesh, J., Shekhar, K., Goldman, M., Tirosh, I., Bialas, A.R., Kamitaki, N., Martersteck, E.M., *et al.*: Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. Cell **161**(5), 1202–1214 (2015)
4. Cao, J., Spielmann, M., Qiu, X., Huang, X., Ibrahim, D.M., Hill, A.J., Zhang, F., Mundlos, S., Christiansen, L., Steemers, F.J., *et al.*: The single-cell transcriptional landscape of mammalian organogenesis. Nature **566**(7745), 496–502 (2019)
5. Tasic, B., Yao, Z., Graybuck, L.T., Smith, K.A., Nguyen, T.N., Bertagnolli, D., Goldy, J., Garren, E., Economo, M.N., Viswanathan, S., *et al.*: Shared and distinct transcriptomic cell types across neocortical areas. Nature **563**(7729), 72–78 (2018)
6. Hochgerner, H., Zeisel, A., Lönnerberg, P., Linnarsson, S.: Conserved properties of dentate gyrus neurogenesis across postnatal development revealed by single-cell rna sequencing. Nature Neuroscience **21**(2), 290–299 (2018)
7. Harris, K.D., Hochgerner, H., Skene, N.G., Magno, L., Katona, L., Gonzales, C.B., Somogyi, P., Kessaris, N., Linnarsson, S., Hjerling-Leffler, J.: Classes and continua of hippocampal ca1 inhibitory neurons revealed by single-cell transcriptomics. PLoS Biology **16**(6), 2006387 (2018)
8. Poličar, P.G., Stražar, M., Zupan, B.: Embedding to reference t-sne space addresses batch effects in single-cell classification. In: International Conference on Discovery Science, pp. 246–260 (2019). Springer
9. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., *et al.*: Scikit-learn: Machine learning in python. the Journal of Machine Learning Research **12**, 2825–2830 (2011)
10. Ulyanov, D.: Multicore-TSNE. GitHub (2016)
11. Linderman, G.C., Rachh, M., Hoskins, J.G., Steinerberger, S., Kluger, Y.: Fast interpolation-based t-sne for improved visualization of single-cell rna-seq data. Nature Methods **16**(3), 243–245 (2019)
12. Jacobs, R.A.: Increased rates of convergence through learning rate adaptation. Neural Networks **1**(4), 295–307 (1988)
13. Belkina, A.C., Ciccolella, C.O., Anno, R., Halpert, R., Spidlen, J., Snyder-Cappione, J.E.: Automated optimized parameters for t-distributed stochastic neighbor embedding improve visualization and analysis of large datasets. Nature Communications **10**(1), 1–12 (2019)
14. Van Der Maaten, L.: Accelerating t-sne using tree-based algorithms. The Journal of Machine Learning Research **15**(1), 3221–3245 (2014)
15. Kobak, D., Berens, P.: The art of using t-sne for single-cell transcriptomics. Nature Communications **10**(1), 1–14 (2019)
16. Kobak, D., Linderman, G., Steinerberger, S., Kluger, Y., Berens, P.: Heavy-tailed kernels reveal a finer cluster structure in t-sne visualisations. arXiv preprint arXiv:1902.05804 (2019)
17. Jacomy, M., Venturini, T., Heymann, S., Bastian, M.: Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. PloS one **9**(6), 98679 (2014)

Figures

Figure 6 Sample figure title. A short description of the figure content should go here.

Figure 7 Sample figure title. Figure legend text.

Tables

Table 1 Sample table title. This is where the description of the table should go.

	B1	B2	B3
A1	0.1	0.2	0.3
A2	...	..	.
A3	..	.	.

Additional Files

Additional file 1 — Sample additional file title
Additional file descriptions text (including details of how to view the file, if it is in a non-standard format or the file extension). This might refer to a multi-page table or a figure.

Additional file 2 — Sample additional file title
Additional file descriptions text.