HHS Public Access

Author manuscript

Nat Methods. Author manuscript; available in PMC 2019 August 11.

Published in final edited form as:

Nat Methods. 2019 March ; 16(3): 243–245. doi:10.1038/s41592-018-0308-4.

Fast Interpolation-based t-SNE for Improved Visualization of Single-Cell RNA-Seq Data

George C. Linderman[1], **Manas Rachh**[1], **Jeremy G. Hoskins**[1], **Stefan Steinerberger**[2], and **Yuval Kluger**[1,3,*]

[1]Applied Mathematics Program, Yale University, New Haven, CT 06511, USA

[2]Department of Mathematics, Yale University, New Haven, CT 06511, USA

[3]Department of Pathology, Yale University School of Medicine, New Haven, CT 06510, USA

Abstract

t-distributed Stochastic Neighborhood Embedding (t-SNE) is widely used for visualizing single-cell RNA-sequencing (scRNA-seq) data, but it scales poorly to large datasets. We dramatically accelerate t-SNE, obviating the need for data downsampling, and hence allowing visualization of rare cell populations. Furthermore, we implement a heatmap-style visualization for scRNA-seq based on one-dimensional t-SNE for simultaneously visualizing the expression patterns of thousands of genes.

1. Main

scRNA-seq enables high-throughput transcriptome profiling at the individual cell level and is increasingly being used to study cell-to-cell heterogeneity in both physiologic and disease processes. Data visualization techniques have played a pivotal role in both analyzing the expression of different marker genes in known cell populations and in identifying new cell types. Over the last decade data visualization using t-SNE has become a cornerstone of scRNA-seq analysis. t-SNE is used to embed a scRNA-seq dataset into a low-dimensional space such that proximal pairs of single cells in the high-dimensional transcriptome space remain proximal in the low dimensional space. The embedding is often colored by the expression levels of a gene of interest, one gene at a time.

Several difficulties arise when applying t-SNE to scRNA-seq data. The number of cells profiled in scRNA-seq experiments has been growing exponentially,[1] with recent datasets measuring the expression of 30,000 genes in over 1,000,000 cells.[2] Profiling such large numbers of cells facilitates the characterization of rare and moderately-sized subpopulations not apparent in smaller samples. However, existing algorithms for constructing t-SNE

*Corresponding author (Yuval.Kluger@yale.edu).

4.Author Contributions

All authors conceived and designed the project. G.C.L. implemented the method. All authors wrote and edited the manuscript.

5.Competing Interests

The authors declare no competing interests.

embeddings are computationally expensive, often necessitating downsampling of the cells prior to running t-SNE, which can in turn result in rare cell populations being missed. Furthermore, removal of the few cells which may express a given marker gene can make even moderately sized populations difficult to identify.

An additional difficulty with applying t-SNE to scRNA-seq data is that overlaying the expression levels of marker genes on separate 2D t-SNE plots is cumbersome owing to the large number of marker genes for each dataset. Practically, only a modest number of such plots can be visually compared.

In this paper, we present two improvements for the application of t-SNE to scRNA-seq data visualization. First, we present FFT-accelerated Interpolation-based t-SNE (FIt-SNE), an algorithm for rapid computation of one- and two-dimensional t-SNE based on polynomial interpolation and further accelerated using the fast Fourier transform. We also present t-SNE heatmaps, a heatmap-style visualization method based on one-dimensional t-SNE, which simultaneously visualizes expression patterns of hundreds to thousands of genes.

FIt-SNE.

t-SNE is often run many times with different parameters and initializations, so that the embedding most consistent with prior knowledge can be chosen. FIt-SNE is a dramatically accelerated implementation of t-SNE, allowing practitioners to analyze entire datasets as opposed to first downsampling. By doing so, FIt-SNE allows practitioners to identify known populations using marker genes which may not be expressed in sufficiently many cells post-downsampling. For example, we used FIt-SNE to embed a dataset consisting of 1.3 million mouse brain cells[2] and identified two known cell types from the Allen Brain Atlas[3] which cannot be identified using a random subset of 50,000 cells (Figure 1), as the latter does not have enough cells expressing both markers. Specifically, GABAergic neurons from the caudal ganglionic eminence which express marker genes Sncg and Slc18a8 and a population of vascular leptomeningeal cells (VLMC) expressing marker genes Spp1 and Col15a1 can both be identified using only the full embedding, as opposed to a random subset.

The t-SNE algorithm solves an optimization problem for embedding the cells (points) in a low-dimensional space based on their transcriptome similarities. Formally, this problem is equivalent to a physical system of particles (points) in which particles exert repulsive and attractive forces on each other. Naively implemented, computing the force each particle exerts on all the other particles is prohibitively slow; we devise approximation schemes for evaluating the repulsive and attractive forces that can scale to millions of points.

Computation of the repulsive forces between every pair of the N points is the most time-consuming step in t-SNE. Instead of calculating the interaction of each point with all the other points (which requires N^2 computations), Barnes-Hut (BH) t-SNE[4] —the fastest published t-SNE implementation—uses a tree structure to compress the interaction between distant cells, hence requiring $N \log N$ computations. We take a different approach by defining a small number p of interpolation nodes, which “mediate” the interaction between the points. First, we calculate the interaction of each point with those nodes ($p \cdot N$ computations). Then we compute the interaction of those nodes with each other (p^2 naively,

$p \log p$ using FFTs). Finally, we interpolate from the interpolation nodes to all of the original points (also $p \cdot N$ computations). Hence, we can approximate the repulsive force in $\sim 2p \cdot N$ computations, as opposed to N^2 or $N \log N$ (Table 1 and S1). We prove rigorous bounds on the approximation error in the Online Methods; in particular, we show that the number of interpolation nodes p required for a certain level of accuracy is independent of N. We set the default FIt-SNE parameters to give an approximation at least as accurate as BH t-SNE's default setting (Figure S1 and Section §8.3.3).

The attractive force between two points decays exponentially fast as a function of the distance between them, so that a point only exerts a significant attractive force on its nearest neighbors. In BH t-SNE, the k–nearest neighbors of each point are identified using vantage-point (VP) trees[5] which tend to be prohibitively expensive for high-dimensional datasets. In FIt-SNE, there are two options for identifying nearest neighbors—multithreaded VP trees and approximate nearest neighbors using ANNOy[6] (Tables 2 and S2). Multithreaded VP trees are exactly as accurate as the VP tree implementation of BH t-SNE, just substantially faster. The use of approximate nearest neighbors is even faster, but could theoretically obscure subtle detail. In practice, however, we find the resulting embedding quality to be essentially indistinguishable (Figures S2, S3, S4, and S5).

Although FIt-SNE makes it practical to run t-SNE on datasets with millions of points, the choice of parameters which lead to an ideal embedding is an active area of research. For example, when the number of points is large, the attractive forces must be exaggerated during the beginning stages of t-SNE in order to ensure optimal embedding of large numbers of points[7] (Supplemental Figure S6). While this paper was in revision, a new paper by Belkina and colleagues (2018)[8] proposed an approach for automatically determining the step size and the optimal number iterations to exaggerate the attractive forces, which they validate using CyTOF and scRNA-seq datasets. In another very recent work, Kobak and Berens (2018)[9] proposed a protocol for exploratory analysis of scRNA-seq data using FIt-SNE (including suggested parameter choices), which leads to dramatically improved embedding quality, particularly with regard to preservation of multi-scale and global structure.

Heatmaps.

Exploration of scRNA-seq data using t-SNE consists of tiling two-dimensional t-SNE plots, each colored by the expression pattern of a different marker gene. Although this information is presented in two dimensions, users are most interested in which genes are associated with which clusters, not the shape or relative locations of the clusters. It has been shown that t-SNE preserved the cluster structure of well-clustered data regardless of the embedding dimension,[7] and thus, one-dimensional t-SNEs usually contain the same information as two-dimensional t- SNEs. Furthermore, multiple one-dimensional t-SNEs, each using different groups of markers, have been previously used to visualize CyTOF data[10] We develop a related approach which exploits the compactness of a single one-dimensional embedding to enable simultaneous exploration of expression patterns of hundreds to thousands of genes in heatmap form. This approach also allows us to discover new marker genes and organize the

genes based on their smoothed expression patterns along the one-dimensional t-SNE representation of the cells.

In t-SNE Heatmaps, we first construct a one-dimensional t-SNE embedding of the cells. Next, we discretize the one-dimensional t-SNE embedding into b bins, where b is user specified, and represent each gene by the sum of its expression in the cells contained in each bin. We then visualize these vectors in heatmap format (i.e. each row is a gene and each column is a bin) using an interactive visualization tool called heatmaply.[12] Notably, unlike dotplots which present the average expression of genes in each cluster (e.g. Figure 2A of Shekhar et al. (2016)[11]), it does not require pre-clustering, and hence can discover patterns in poorly clustered data that might be missed if averaging across clusters.

Various strategies can be used to select the genes presented in the heatmap. If the user has prior knowledge as to genes of interest, these genes can be presented, along with genes whose onedimensional t-SNE binned representation are most similar, allowing for marker gene discovery. If the user wants to identify genes specific to clusters, a “metagene” can be constructed, which is 1 on cells in a cluster and 0 elsewhere. Then genes whose one-dimensional t-SNE binned representation are most similar to these “metagenes” (ie. specific to a cluster) can be presented in the heatmap. “Metagenes” for combinations of clusters can also be constructed.

Figure 2 demonstrates t-SNE heatmaps using retinal bipolar cells from Shekhar et al. (2016).[11] In this work, scRNA-seq was used to profile ~ 25,000 mouse retinal bipolar cells and classify them into 15 types. Using graph-based clustering techniques, cells were clustered, and marker genes corresponding to each of the putative subtypes of bipolar cells were subsequently identified. We embedded these bipolar cells using 1D t-SNE and found the 25 genes most associated with the marker genes listed in Table S2 of Shekhar et al. (2016). We also found the 25 genes most associated with “metagenes” for each cluster in the 2D t-SNE. The resulting t-SNE heatmap (Figure 2, Supplementary Figures S7, and S8) identified all 16 of the new bipolar cell markers listed in Figure 2A of Shekhar et al. (2016). The clustered structure of the dataset is evident in the heatmap, and the user can zoom in to identify the genes that characterize and distinguish different regions of the embedding. We note that the structure is substantially clearer than a heatmap of the same genes binned using standard hierarchical clustering, even when the rows are ordered as in the t-SNE heatmaps (Figure S9).

2. Methods

R, Python, and Matlab implementations of FIt-SNE and an R implementation of t-SNE heatmaps are available from https://github.com/KlugerLab/. Methods, including statements of data availability and any associated accession codes and references, are available in the online version of the paper. The Life Sciences Reporting Summary was also completed.

8. Online Methods

We first briefly review the t-SNE approach and then then present FIt-SNE’s method for optimizing the computation of the repulsive force in Section §8.3. Section §8.4 presents an

implementation of out-of-core PCA for the analysis of datasets too large to fit in the memory. Finally, Section §8.5 provides details of the embedding of 1.3 million mouse brain cells (Figure 1), Section §8.6 describes the demonstration of t-SNE heatmaps (Figure 2), and Section §8.7 provides details about our comparison of VP trees to approximate nearest neighbors on three scRNA-seq datasets.

8.1. t-distributed Stochastic Neighborhood Embedding.

Given a d-dimensional dataset $X = \{x_1, x_2, \ldots, x_N\} \subset \mathbb{R}^d$, t-SNE aims to compute the low-dimensional embedding

$$Y = \{y_1, y_2, \ldots, y_N\} \subset \mathbb{R}^s,$$

where $s \ll d$, such that if two points x_i and x_j are close in the input space, then their corresponding points y_i and y_j are also close. Affinities between points x_i and x_j in the input space, p_{ij}, are defined as

$$p_{i \mid j} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)} \quad \text{and} \quad p_{ij} = \frac{p_{i \mid j} + p_{j \mid i}}{2N}.$$

Here σ_i is the bandwidth of the Gaussian distribution is computed based on the user-specified perplexity P_i (the conditional distribution of all other points given x_i). Similarly, the affinity between points y_i and y_j in the embedding space is defined using the Cauchy kernel

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}.$$

t-SNE finds the points $\{y_1, \ldots, y_n\}$ that minimize the Kullback-Leibler divergence between the joint distribution of points in the input space P and the joint distribution of the points in the embedding space Q,

$$C(\mathcal{Y}) = KL(P \| Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

Starting with a random initialization, the cost function $C(\mathcal{Y})$ is minimized by gradient descent, with the gradient[13]

$$\frac{\partial C}{\partial y_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij}) q_{ij} Z (y_i - y_j),$$

where Z is a global normalization constant

$$Z = \sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}.$$

We split the gradient into two parts

$$\frac{1}{4}\frac{\partial C}{\partial y_i} = \sum_{j \neq i} p_{ij} q_{ij} Z(y_i - y_j) - \sum_{j \neq i} q_{ij}^2 Z(y_i - y_j)$$

where the first sum $F_{\text{attr},i}$ corresponds to an attractive force between points and the second sum $F_{\text{rep},i}$ corresponds to a repulsive force

$$\frac{1}{4}\frac{\partial C}{\partial y_i} = F_{\text{attr},i} - F_{\text{rep},i}.$$

The computation of the gradient at each step is an N-body simulation, where the position of each point is determined by the forces exerted on it by all other points. Exact computation of N-body simulations scales as $O(N^2)$, making exact t-SNE computationally prohibitive for datasets with tens of thousands of points. It should be noted that since the input similarities do not change they can be precomputed and hence do not dominate the computational time.

8.2. Early Exaggeration.

In the expression for the gradient descent, the sum of attractive and repulsive forces,

$$\frac{1}{4}\frac{\partial C}{\partial y_i} = \alpha \sum_{j \neq i} p_{ij} q_{ij} Z(y_i - y_j) - \sum_{j \neq i} q_{ij}^2 Z(y_i - y_j),$$

the numerical quantity $\alpha > 0$ plays a substantial role as it determines the strength of attraction between points that are similar (in the sense of pairs x_i, x_j with p_{ij} large). In early exaggeration, first α =12 for the first several hundred iterations, after which it set[13] to 1. One of the main results of Linderman and Steinerberger (2017)[7] is that α plays a crucial role and that when it is set large enough, t-SNE is guaranteed to separate well-clustered data and also successfully embed various synthetic datasets (e.g. a swiss roll) that were previously thought to be poorly embedded by t-SNE.

8.3. Accelerating computation of repulsive forces in FIt-SNE.

In existing methods, the repulsive forces $F_{\text{rep},i}$ are approximated at each iteration using the Barnes-Hut Algorithm,[17] a tree-based algorithm which scales as $O(N \log N)$, where N is the total number of data points. In this work, we present an interpolation-based fast Fourier transform accelerated algorithm for computing $F_{\text{repul},i}$ which scales as $O(N)$. Moreover, empirical tests show a significant improvement over the Barnes-Hut approach for any sized system.

Recall that, $\{y_1, y_2, \ldots, y_N\}$ is the s-dimensional embedding of a collection of d-dimensional vectors $\{x_1, \ldots, x_N\}$. At each step of gradient descent, the repulsive forces are given by

$$F_{\text{rep},k}(m) = \left(\sum_{\substack{\ell=1 \\ \ell \neq k}}^{N} \frac{y_\ell(m) - y_k(m)}{(1 + \|y_\ell - y_k\|^2)^2} \right) \Bigg/ \left(\sum_{j=1}^{N} \sum_{\substack{\ell=1 \\ \ell \neq j}}^{N} \frac{1}{(1 + \|y_\ell - y_j\|^2)} \right), \quad (1)$$

where $k = 1, 2, \ldots N$, $m = 1, 2 \ldots s$, and $y_i(j)$ denotes the j^{th} component of y_i. Evidently, the repulsive force between the vectors $\{y_1, \ldots, y_N\}$ consists of N^2 pairwise interactions, and were it computed directly, would require CPU-time scaling as $O(N^2)$. Even for datasets consisting of a few thousand points, this cost becomes prohibitively expensive. Our approach enables the accurate computation of these pairwise interactions in $O(N)$ time. Since the majority of applications of t-SNE are for at most two-dimensional embeddings, in the following we focus our attention on the cases where $s = 1$ or 2. However, we note that our algorithm extends naturally to arbitrary dimensions. In such cases, though the constants in the computational cost will vary, our approach will still yield an algorithm with a CPU-time which scales as $O(N)$.

We begin by observing that the repulsive forces $F_{\text{rep},k}$ defined in eq. (1) can be expressed as $s + 2$ sums of the form

$$\phi(y_i) = \sum_{j=1}^{N} K(y_i, y_j) q_j \quad (2)$$

where the kernel $K(y, z)$ is either

$$K_1(y, z) = \frac{1}{(1 + \|y - z\|^2)}, \quad \text{or} \quad K_2(y, z) = \frac{1}{(1 + \|y - z\|^2)^2}, \quad (3)$$

for $y, z \in \mathbb{R}^s$. Note that both of the kernels K_1 and K_2 are smooth functions of y, z for all $y, z \in \mathbb{R}^s$. The key idea of our approach is to use polynomial interpolants of the kernel K in order to accelerate the evaluation of the N–body interactions defined in eq. (2).

8.3.1. Mathematical Preliminaries.—First, we demonstrate with a simple example how polynomial interpolation can be used to accelerate the computation of the N–body interactions with a smooth kernel. Suppose that $y_1, \ldots, y_M \in (y_0, y_0 + R)$ and $z_1, \ldots, z_N \in (z_0, z_0 + R)$. Let I_{y_0} and I_{z_0} denote the intervals $(y_0, y_0 + R)$ and $(z_0, z_0 + R)$, respectively. Note that no assumptions are made regarding the relative locations of y_0 and z_0; in particular, the case $y_0 = z_0$ is also permitted.

Now consider the sums

$$\phi(y_i) = \sum_{j=1}^{N} K(y_i, z_j) q_j, \quad i = 1, 2, \ldots M. \quad (4)$$

Let p be a positive integer. Suppose that $\tilde{z}_1, \ldots, \tilde{z}_p$, are a collection of p points on the interval I_{z0} and that $\tilde{y}_1, \ldots, \tilde{y}_p$, are a collection of p points on the interval I_{y0}. Let $K_p(y, z)$ denote a bivariate polynomial interpolant of the kernel $K(y, z)$ satisfying

$$K_p(\tilde{y}_j, \tilde{z}_\ell) = K(\tilde{y}_j, \tilde{z}_\ell), \quad j, \ell = 1, 2, \ldots p.$$

A simple calculation shows that $K_p(y, z)$ is given by

$$K_p(y, z) = \sum_{\ell=1}^{p} \sum_{j=1}^{p} K(\tilde{y}_j, \tilde{z}_\ell) L_{j,\tilde{y}}(y) L_{\ell,\tilde{z}}(z), \quad (5)$$

where $L_{j,\tilde{y}}(y)$ and $L_{\ell,\tilde{z}}(z)$ are the Lagrange polynomials

$$L_{\ell,\tilde{y}}(y) = \prod_{\substack{j=1 \\ j \neq \ell}}^{p} (y - \tilde{y}_j) \Big/ \prod_{\substack{j=1 \\ j \neq \ell}}^{p} (\tilde{y}_\ell - \tilde{y}_j), \quad \text{and} \quad L_{\ell,\tilde{z}}(z) = \prod_{\substack{j=1 \\ j \neq \ell}}^{p} (z - \tilde{z}_j) \Big/ \prod_{\substack{j=1 \\ j \neq \ell}}^{p} (\tilde{z}_\ell - \tilde{z}_j),$$

ℓ=1, 2 … p. In the following we will refer to the points $\tilde{y}_1, \ldots, \tilde{y}_p$, and $\tilde{z}_1, \ldots, \tilde{z}_p$ as interpolation points.

Let $\tilde{\phi}(y_i)$ denote the approximation to $\phi(y_i)$ obtained by replacing the kernel K in eq. (4) by its polynomial interpolant K_p, i.e.

$$\tilde{\phi}(y_i) = \sum_{j=1}^{N} K_p(y_i, z_j) q_j,$$

for i = 1, 2 … M. Clearly the error in approximating $\phi(y_i)$ via $\tilde{\phi}(y_i)$ is bounded (up to a constant) by the error in approximating $K(y, z)$ via $K_p(y, z)$. In particular, if the polynomial interpolant satisfies the inequality

$$\sup_{\substack{y \in (y_0, y_0 + R) \\ z \in (z_0, z_0 + R)}} | K_p(y, z) - K(y, z) | \leq \varepsilon, \quad (6)$$

then the error $| \tilde{\phi}(y_i) - \phi(y_i) |$ is given by

$$
\begin{aligned}
|\tilde{\phi}(y_i) - \phi(y_i)| &= \left| \sum_{j=1}^{N} (K_p(y_i, z_j) - K(y_i, z_j)) q_j \right| \\
&\leq \sum_{j=1}^{N} |K_p(y_i, z_j) - K(y_i, z_j)| \, |q_j| \\
&\leq \varepsilon \sum_{j=1}^{N} |q_j| \, .
\end{aligned}
$$

A direct computation of $\phi(y_1), \ldots, \phi(y_M)$ requires $O(M \cdot N)$ operations. On the other hand, the values $\tilde{\phi}(y_i)$, $i = 1, 2, \ldots M$, can be computed in $O((M + N) \cdot p + p^2)$ operations as follows. Using eq. (5), $\tilde{\phi}(y_i)$ can be rewritten as

$$
\begin{aligned}
\tilde{\phi}(y_i) &= \sum_{j=1}^{N} \sum_{\ell=1}^{p} \sum_{m=1}^{p} K(\tilde{y}_\ell, \tilde{z}_m) L_{\ell, \tilde{y}}(y_i) L_{m, \tilde{z}}(z_j) q_j, \\
&= \sum_{\ell=1}^{p} L_{\ell, \tilde{y}}(y_i) \left(\sum_{m=1}^{p} K(\tilde{y}_\ell, \tilde{z}_m) \left(\sum_{j=1}^{N} L_{m, \tilde{z}}(z_j) q_j \right) \right),
\end{aligned}
$$

for $i = 1, 2, \ldots M$. The values $\tilde{\phi}(y_1), \ldots, \tilde{\phi}(y_M)$, are computed in three steps.

- **Step 1**: Compute the coefficients w_m defined by the formula

  $$w_m = \sum_{j=1}^{N} L_{m, \tilde{z}}(z_j) q_j,$$

  for each $m = 1, 2, \ldots p$. This step requires $O(N \cdot p)$ operations.

- **Step 2**: Compute the values v_ℓ at the interpolation nodes $\tilde{y}_\ell$ defined by the formula

  $$v_\ell = \sum_{m=1}^{p} K(\tilde{y}_\ell, \tilde{z}_m) w_m$$

  for all $\ell = 1, 2, \ldots p$. This step requires $O(p^2)$ operations.

- **Step 3**: Evaluate the potential $\tilde{\phi}(y_i)$ using the formula

  $$\tilde{\phi}(y_i) = \sum_{\ell=1}^{p} L_{\ell, \tilde{y}}(y_i) v_\ell,$$

  for all $i = 1, 2 \ldots M$. This step requires $O(M \cdot p)$ operations.

See Figure S10 for an illustrative figure of the above procedure.

8.3.2. Algorithm.—In this section, we present the main algorithm for the rapid evaluation of the repulsion forces eq. (2). The central strategy is to use piecewise polynomial interpolants of the kernel with equispaced points, and use the procedure described in Section §8.3.1.

Specifically, suppose that the points y_i, $i = 1, 2, \dots N$ are all contained in the interval $[y_{\min}, y_{\max}]$. We subdivide the interval $[y_{\min}, y_{\max}] = \bigcup_{i=1}^{N_{\text{int}}} I_j$, into N_{int} intervals of equal length. Let $\tilde{y}_{j,\ell}$ denote p equispaced nodes on the interval I_l given by

$$\tilde{y}_{j,\ell} = h/2 + ((j-1) + (\ell - 1) \cdot p) \cdot h, \quad (7)$$

where $h = 1/(N_{\text{int}} \cdot p)$, $j = 1, 2 \dots p$, and $\ell = 1, 2 \dots N_{\text{int}}$.

Remark 1. The nodes $\tilde{y}_{j,\ell}$, $j = 1, 2 \dots p$, and $\ell = 1, 2, \dots N_{\text{int}}$, defined in eq. (7), are also equispaced on the whole interval $[y_{\min}, y_{\max}]$.

The interaction between any two intervals I, J, i.e.

$$\sum_{y_j \in J} K(y_i, y_j) q_j, \quad y_i \in I$$

can be accelerated via the algorithm discussed in section 8.3.1. This procedure amounts to using a piecewise polynomial interpolant of the kernel $K(y, z)$ on the domain $y, z \in [y_{\min}, y_{\max}]$ as opposed to using an interpolant on the whole interval. We summarize the procedure below.

- **Step 1**: For each interval I_ℓ $\ell = 1, 2, \dots N_{\text{int}}$, compute the coefficients $w_{m,\ell}$ defined by the formula

$$w_{m,\ell} = \sum_{y_j \in I_\ell} L_{m,\tilde{y}^\ell}(y_j) q_j,$$

  for each $m = 1, 2, \dots p$. This step requires $O(N \cdot p)$ operations.

- **Step 2**: Compute the values $v_{m,n}$ at the equispaced nodes $\tilde{y}_{m,n}$ defined by the formula

$$v_{m,n} = \sum_{j=1}^{N_{\text{int}}} \sum_{\ell=1}^{p} K(\tilde{y}_{m,n}, \tilde{y}_{\ell,j}) w_{\ell,j} \quad (8)$$

  for all $m = 1, 2, \dots p$, $n = 1, 2 \dots N_{\text{int}}$. This step requires $O((N_{\text{int}} \cdot p)^2)$ operations.

- **Step 3**: For each interval I_ℓ ℓ=1, 2, … N_{int}, compute the potential $\phi(y_i)$ via the formula

$$\phi(y_i) = \sum_{j=1}^{p} L_{j,\tilde{y}^\ell}(y_i) v_{j,\ell},$$

for all points $y_i \in I_\ell$ This step requires $O(N \cdot p)$ operations.

In this procedure, the functions $L_{j,\tilde{y}^\ell}, j = 1, 2, \ldots p$, are the Lagrange polynomials corresponding to the equispaced interpolation nodes on interval I_ℓ

In Step 2 of the above procedure, we are evaluating N–body interactions on equispaced grid points. For notational convenience, we rewrite the sum eq. (8)

$$v_i = \sum_{j=1}^{N_{\text{int}} \cdot p} K(\tilde{y}_i, \tilde{y}_j) w_j, \quad (9)$$

$i = 1, 2, \ldots N_{\text{int}} \cdot p$. The kernels of interest (K_1 and K_2 defined in eq. (3)) are translationally-invariant, i.e., the kernels satisfy $K(y, z) = K(y + \delta, z + \delta)$ for any δ. The combination of using equispaced points, along with the translational-invariance of the kernel, implies that the matrix associated with the evaluation of the sums eq. (9) is Toeplitz. This computation can thus be accelerated via the fast-Fourier transform (FFT), which reduces the computational complexity of evaluating the sums eq. (9) from $O((N_{\text{int}} \cdot p)^2)$ operations to $O(N_{\text{int}} \cdot p \log (N_{\text{int}} \cdot p))$.

Algorithm 1 describes the fast algorithm for evaluating the repulsive forces eq. (2) in one dimension (s=1) which has computational complexity $O(N \cdot p + (N_{\text{int}} \cdot p) \log (N_{\text{int}} \cdot p))$.

Algorithm 1: FFT-accelerated Interpolation-based t-SNE (FIt-SNE)

Input: Collection of points $\{y_i\}_{i=1}^N$, source strengths $\{q_i\}_{i=1}^N$, number of intervals N_{int}, number of interpolation points per interval p

Output: $\phi(y_i) = \sum_{j=1}^N K(y_i, y_j) q_j$ for $i = 1, 2, \ldots N$

1 For each interval I_ℓ, form the equispaced nodes $\tilde{y}_{j,\ell}$, $j = 1, 2, \ldots p$ given by eq. (7)

2 **for** $I \leftarrow 1$ **to** N_{int} **do**

3 Compute the coefficients $w_{m,\ell}$ given by

$$w_{m,\ell} = \sum_{y_i \in I_\ell} L_{m,\tilde{y}^\ell}(y_i) q_i,$$

$m = 1, 2, \ldots p$.

4 **end**

5 Use the fast-Fourier transform to compute the values of $v_{m,n}$ given by

$$\begin{bmatrix} v_{1,1} \\ v_{2,1} \\ \vdots \\ v_{p-1,N_{\text{int}}} \\ v_{p,N_{\text{int}}} \end{bmatrix} = \tilde{K} \cdot \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{p-1,N_{\text{int}}} \\ w_{p,N_{\text{int}}} \end{bmatrix}, \tag{10}$$

where $\tilde{K}$ is the Toeplitz matrix given by

$$\tilde{K}_{i,j} = K(\tilde{y}_i, \tilde{y}_j), \tag{11}$$

$i, j = 1, 2, \ldots N_{\text{int}} \cdot p$.

6 **for** $I \leftarrow 1$ **to** N_{int} **do**

7 Compute $\phi(y_i)$ at all points $y_i \in I_\ell$ via

$$\phi(y_i) = \sum_{j=1}^p L_{j,\tilde{y}^\ell}(y_i) v_{j,\ell}$$

8 **end**

8.3.3. Optimal choice of p and N_{int}.— Recall that the computational complexity of Algorithm 1 is $O(N \cdot p + N_{\text{int}} \cdot p \log(N_{\text{int}} \cdot p))$. We remark that the choice of the parameters N_{int} and p depends solely on the specified tolerance ε and is independent of the number of points N. Generally, increasing p will reduce the number of intervals N_{int} required to obtain the same accuracy in the computation. However, we observe that the reduction in N_{int} for an increased p is not advantageous from a computational perspective—since, as the number of points N increases, the computational cost is independent of N_{int} and is only a function of p. Moreover, for the t-SNE kernels K_1 and K_2 defined in eq. (3), it turns out that for a fixed accuracy the product $N_{\text{int}} \cdot p$ remains nearly constant for $p \geq 3$. Thus, it is optimal to use $p = 3$ for all t-SNE calculations. In a more general environment, when higher accuracy is required and for other translationally invariant kernels K, the choice of the number of nodes

per interval p and the total number of intervals N_{int} can be optimized based on the accuracy of computation required.

Remark 2. Special care must be taken when increasing p in order to achieve higher accuracy due to the Runge phenomenon associated with equispaced nodes. In fact, the kernels that arise in t-SNE are archetypical examples of this phenomenon. Since we use only low-order piecewise polynomial interpolation ($p = 3$), we encounter no such difficulties.

In our simulations, we set the values of $p = 3$ and $N_{\text{int}} = \max(50, \lceil y_{max} - y_{min} \rceil)$. These values are chosen to ensure that the computation of $F_{rep,i}$ is at least as accurate as the Barnes-Hut approximation at default setting ($\theta = 0.5$). We test the accuracy of the two methods by comparing the repulsive forces computed using BH t-SNE and FIt-SNE to the exact repulsive forces computed using direct algorithm on a dataset with 4000 points. In Figure S1, we report the relative error of the BH t-SNE and FIt-SNE approximations at default values and note that the latter achieves the same (or better) accuracy. Since the approximation error is independent of the number of points (Section §8.3.6), this error analysis applies to datasets of any size.

8.3.4. Extension to two dimensions.—The above algorithm naturally extends to two-dimensional embeddings (s=2). In this case, we divide the computational square $[y_{\min}, y_{\max}] \times [y_{\min}, y_{\max}]$ into a collection of $N_{\text{int}} \times N_{\text{int}}$ squares with equal side length, and for polynomial interpolation, we use tensor product $p \times p$ equispaced nodes on each square. The matrix $\tilde{K}$ mapping the coefficients w to the coefficients v which is of size $(N_{\text{int}} \cdot p)^2 \times (N_{\text{int}} \cdot p)^2$, is not a Toeplitz matrix, however, it can be embedded into a Toeplitz matrix of twice its size. The computational complexity of the algorithm analogous to Algorithm 1 for two-dimensional t-SNE is $O(N \cdot p^2 + (N_{\text{int}} \cdot p)^2 \log (N_{\text{int}} \cdot p))$.

8.3.5. Performance comparison.—The datasets for comparing the CPU-time performance of BH t-SNE and FIt-SNE in Tables 1, 2, S1, and S2 are generated in the following manner. For each N, we sample $N/10$ points from 10 gaussians in d–dimensions with mean $c_j \in \mathbb{R}^d$ and fixed variance $\sigma = 0.0001$. The experiments were performed on two systems—a 2017 Macbook Pro laptop with 2.9 GHz (Turbo up to 3.6GHz) Intel i7 CPU with 2 cores (each supporting 4 threads) and 16GB RAM; and a server with Intel Xeon CPUs with 24 cores clocked at 2.4 GHz and 500GB RAM. In FIt-SNE, the computation of nearest neighbors when computing input similarities, the summing of attractive forces at each iteration of gradient descent, and step 3 of the interpolation scheme outlined above are all multithreaded using C++11 threads, whereas the rest of the computation of the repulsive forces is done via single thread FFTs owing to the small size of FFTs involved. The poorer performance of both BH t-SNE and FIt-SNE on the server as compared to the Macbook can be attributed to the slower single processor clock speed.

8.3.6. Approximation error estimates.—In this section we prove error estimates related to interpolation by equispaced points on a subinterval of the computational domain. First we fix x_0 and suppose that $K(x_0, y)$ is to be approximated on the interval $[a, b]$ by the p-point Lagrange inter-polant $w_p(y)$. For ease of exposition, let $f(y) = K(x_0, y)$ where $K(x,$

y) is either K_1 or K_2 given by eq. (3). Then, a classical theorem in approximation theory (see Dalquist and Björck (2008)[18] for example) states that for all $y \in (a, b)$ there exists a $\zeta_y \in (a, b)$ such that

$$E_p(y) = f(y) - w_p(y) = \frac{f^{(p)}(\zeta_y)}{p!}\pi_p(y),$$

where $f^{(p)}$ denotes the pth derivative of f, and

$$\pi_p(y) = \prod_{k=1}^{p}(y - y_j).$$

Let $h = (b - a)/p$ and the interpolation nodes on the interval (a, b) are $y_j = a + (j - 1/2)h$, j = 1, …, p.

We bound $\pi_p(y)$ in the following way (see Trefethen (2013)[19] for example). Suppose that $y_j < y < y_{j+1}$. Then

$$\begin{aligned} |\pi_p(y)| &= |y - y_1| \cdot |y - y_2| \cdots |y - y_p| \\ &\le hj \cdots 2h\,(y - y_j)(y_{j+1} - y)2h \cdot 3h \cdots (p - j)h \\ &= h^{p-2} j!\,(p - j)!(y - y_j)(y_{j+1} - y) \\ &= \frac{h^p j!(p - j)!}{4}. \end{aligned}$$

Clearly this is bounded by $\frac{h^p(p-1)!}{4}$. Similarly, if $y < y_1$, or $y > y_p$ then

$$|\pi_p(y)| \le \frac{h}{2}\frac{3h}{2}\cdots\frac{2p-1}{2} = \frac{(2p)!}{2^{2p}p!}h^p.$$

In order to bound $f^{(p)}(\zeta_y)$ we first consider the case where $f(y) = K_1(x_0, y)$. Then

$$f(y) = \frac{1}{1 + \|y - x_0\|^2} = \frac{1/2}{1 + i(y - x_0)} + \frac{1/2}{1 - i(y - x_0)}.$$

Taking p derivatives we obtain

$$f^{(p)}(y) = \frac{1}{2}p!\,i^p\left[\frac{(-1)^p}{[1 + i(y - x_0)]^p} + \frac{1}{[1 - i(y - x_0)]^p}\right]$$

and hence

$$| f^{(p)}(y) | \leq p!$$

Similarly, if $f(y) = K_2(x_0, y)$ then

$$f(y) = \frac{1}{(1 + \|y - z\|^2)^2} = \frac{1/4}{[1 + i(y - x_0)]^2} + \frac{1/4}{[1 - i(y - x_0)]^2} - \frac{1/4}{1 + i(y - x_0)} - \frac{1/4}{1 - i(y - x_0)},$$

from which it follows that

$$| f^{(p)}(y) | \leq \frac{p+2}{2} p!.$$

Putting the above estimates together gives

$$| E_p(y) | \leq \frac{(2p)!}{2^{2p} p!} h^p \frac{p+2}{2} = \frac{(2p)!}{2^{2p} p!} (b-a)^p \frac{1}{p^p} \frac{p+2}{2},$$

which holds for both K_1 and K_2. Using Stirling's approximation (see Abramowitz and Stegun (1965),[20] for example) it follows that

$$| E_p(y) | \leq \left(\frac{p+2}{\sqrt{2}}\right)\left(\frac{b-a}{e}\right)^p e^{\frac{1}{24p}}.$$

We now use this estimate to construct an error bound of the form given in eq. (6). First, for fixed $x \in [a, b]$ let $K_r(x, y)$ denote the polynomial interpolant for $y \in [c, d]$. Then

$$\max_{x \in [a,b]} \max_{y \in [c,d]} | K(x, y) - K_r(x, y) | \leq \left(\frac{p+2}{\sqrt{2}}\right)\left(\frac{d-c}{e}\right)^p e^{\frac{1}{24p}}.$$

Similarly, for fixed $y \in [c, d]$ let $K_\ell(x, y)$ denote the polynomial interpolant for $x \in [a, b]$, in which case

$$\max_{x \in [a,b]} \max_{y \in [c,d]} | K(x, y) - K_\ell(x, y) | \leq \left(\frac{p+2}{\sqrt{2}}\right)\left(\frac{d-c}{e}\right)^p e^{\frac{1}{24p}}.$$

Note that by construction,

$$K_r(x, y) = \sum_{j=1}^{p} L_{j,[c,d]}(y) K(x, y_j),$$

and

$$K_\ell(x,y) = \sum_{j=1}^{p} L_{j,[a,b]}(x) K(x_j, y),$$

where $L_{j,[c,d]}$, $j = 1, \ldots, p$ are the Lagrange polynomials for the nodes $y_1, \ldots, y_p \in [c, d]$.

As above, let $K_p(x, y)$ denote the polynomial interpolant of $K(x, y)$ which is degree p in both x and y for $x \in [a, b]$ and $y \in [c, d]$. Evidently,

$$K_p(x,y) = \sum_{j=1}^{p} \sum_{m=1}^{p} L_{j,[c,d]}(y) L_{m,[a,b]}(x) K(x_m, y_j).$$

Hence

$$\max_{x\in[a,b]} \max_{y\in[c,d]} |K_p(x,y) - K_r(x,y)| \le \max_{x\in[a,b]} \max_{y\in[c,d]} \sum_{j=1}^{p} \left| L_{j,[c,d]}(y) \right| \left| K(x,y_j) - \sum_{m=1}^{p} L_{m,[a,b]}(x) K(x_m, y_j) \right|$$

$$= \max_{x\in[a,b]} \max_{y\in[c,d]} \sum_{j=1}^{p} \left| L_{j,[c,d]}(y) \right| |K(x,y_j) - K_\ell(x,y_j)|$$

$$\le \left(\frac{p+2}{\sqrt{2}}\right)\left(\frac{b-a}{e}\right)^p e^{\frac{1}{24p}} \sum_{j=1}^{p} \max_{y\in[c,d]} |L_{j,[c,d]}(y)|.$$

A slight modification of the argument presented in Trefethen and Weideman (1991)[21] yields the following bound,

$$\max_{y\in[c,d]} |L_{j,[c,d]}(y)| \le 8\frac{2^p}{p},$$

from which it follows that

$$\max_{x\in[a,b]} \max_{y\in[c,d]} |K_p(x,y) - K_r(x,y)| \le 8\left(\frac{p+2}{\sqrt{2}p}\right)\left(\frac{2(b-a)}{e}\right)^p e^{\frac{1}{12p}}.$$

Then

$$|K(x_0,y_0) - K_p(x_0,y_0)| \le |K(x_0,y_0) - K_r(x_0,y_0)| + |K_r(x_0,y_0) - K_p(x_0,y_0)|$$

$$\le 8\left(\frac{p+2}{\sqrt{2}p}\right)\left(\frac{2(b-a)}{e}\right)^p e^{\frac{1}{12p}} + \left(\frac{p+2}{\sqrt{2}}\right)\left(\frac{d-c}{e}\right)^p e^{\frac{1}{24p}}$$

which is the estimate we require. In particular, if $L = b - a = d - c$ we obtain the bound

$$| K(x_0, y_0) - K_p(x_0, y_0) | \leq 7\frac{(p+2)}{p}\frac{2^p L^p}{e^p}.$$

Note that if $L < \frac{e}{2}$ then the error will decay exponentially in p.

In two-dimensions an almost identical analysis shows that the error is bounded by

$$| K(x_0, y_0) - K_p(x_0, y_0) | \leq 16^3\frac{(p+2)}{\sqrt{8}p^3}\frac{8^p L^p}{e^p}.$$

In principle this guarantees convergence only when $L < \frac{e}{8}$. In practice, extensive numerical evidence suggests that the error decays exponentially in p provided that $L < 1.4$.

8.4. Out-of-Core PCA.

The methods for t-SNE presented above allows for the embedding of millions of points, but can only be used to reduce the dimensionality of datasets that can fit in the memory. For many large, high dimensional datasets, specialized servers must be used simply in order to load the data. In order to allow for visualization and analysis of such datasets on resource-limited machines, we present an out-of-core implementation of randomized PCA, which can be used to compute the top few (e.g. 50) principal components of a dataset to high accuracy, without ever loading it in its entirety.[22] Note that out-of-core PCA was not used in the analysis above, but we include it as it can be useful for users interested in running t-SNE on large datasets using a resource-limited machine.

8.4.1. Randomized Methods for PCA.—The goal of PCA is to approximate the matrix being analyzed (after mean centering of its columns) with a low-rank matrix. PCA is primarily useful when such an approximation makes sense; that is, when the matrix being analyzed is approximately low-rank. If the input matrix is low-rank, then by definition, its range is low-dimensional. As such, when the input matrix is applied to a small number of random vectors, the resulting vectors nearly span its range. This observation is the core idea behind randomized algorithms for PCA: applying the input matrix to a small number of random vectors results in vectors that approximate the range of the matrix. Then, simple linear algebra techniques can be used to compute the principal components. Notably, the only operations involving the large input matrix are matrix-vector multiplications, which are easily parallelized, and for which highly optimized implementations exist. Randomized algorithms have been rigorously proven to be remarkably accurate with extremely high probability,[25,26] because for a rank-k matrix, as few as $l = k + 2$ random vectors are sufficient for the probability of missing a significant part of the range to be negligible. The algorithm and its underlying theory are covered in detail in Halko et al. (2011).[25] An easy-to-use “black box” implementation of randomized PCA is available and described in Li et al. (2017),[23] but it requires the entire matrix to be loaded in the memory. We present an out-of-core implementation of PCA in C++/R, oocPCA, allowing for decomposition of matrices which cannot fit in the memory.

Algorithm 2: Out-of-Core PCA (oocPCA)

Input: Matrix A of size $m \times n$ stored in slow memory, non-negative integers its, k, l, b, where $0 < k \leq l < \min(m, n)$, and l defaults to $k + 2$

Output: Orthonormal U of size $m \times k$, non-negative diagonal matrix Σ of size $k \times k$, orthonormal V of size $n \times k$, such that $A \approx U\Sigma V^*$

Generate uniform random matrix Ω of size $n \times l$

Form $Y_0 = A\Omega$ block-wise, b rows at a time

Renormalize with LU factorization $L_0 U_0 = Y_0$

for $i \leftarrow 1$ **to** its **do**

  From $Y_i = AA^* L_{i-1}$ block-wise, b rows at a time

  if $i < its$ **then**

    Renormalize with LU factorization $L_i U_i = Y_i$

  end

end

Renormalize with QR factorization $QR = Y_i$

Compute SVD of small matrix $U'\Sigma V^* = Q^* A$

Set $U = QU'$

8.4.2. Implementation.—Our implementation is described in Algorithm 1. Given an $m \times n$ matrix of doubles A, stored in row-major format on the disk of a machine with M bytes of available memory, the number of rows that can fit in the memory is calculated as $b = \left\lfloor \frac{M}{8mn} \right\rfloor$. The only operations performed using A are matrix multiplications, which can be performed block-wise. Specifically, the matrix product AB, where B is an $n \times p$ matrix stored in the fast memory, can be computed by loading the first b rows of A, and forming the inner product of each row with the columns of B. The process can be continued with the remaining blocks of the matrix, essentially "filling in" the product AB with each new block. In this manner, left multiplication by A can be computed without ever loading the full matrix A.

By simply replacing the matrix multiplications in the implementation of Li et al. (2017)[23] with block-wise matrix multiplication, an out-of-core algorithm can be obtained. However, significant optimization is possible. The run-time of an out-of-core algorithm is almost entirely determined by disk access time; namely, the number of times the matrix must be loaded to the memory. As suggested in Li et al. (2017),[23] the renormalization step between the application of A and A^* is not necessary in most cases, and in the out-of-core setting, doubles the number of times A must be loaded per power iterations. In our implementation, we remove this renormalization step, and apply AA^* simultaneously, hence requiring the matrix only be loaded once per iteration.

Our implementation is in C++ with an R wrapper. For maximum optimization of linear algebra operations, we use the highly parallelized Intel MKL for all BLAS functions (e.g.

matrix multiplications). The R wrapper provides functions for PCA of matrices in CSV and in binary format. Furthermore, basic preprocessing steps including log transformation and mean centering of rows and/or columns can also be performed prior to decomposition, so that the matrix need not ever be fully stored in the memory.

To demonstrate oocPCA's performance, we generated a random 1,000,000 × 30,000 rank-50 matrix stored as doubles, which would require 240GB to simply store in the memory, far exceeding the memory capacity of a personal computer. Using oocPCA we can compute the top principal components of the matrix with much less memory. Using a 2017 Macbook Pro laptop with 16GB RAM, solid state drive, and a 2.9 GHz Intel i7 CPU, the rank-50 approximation was computed in 38 minutes.

8.5. FIt-SNE of 1.3 million mouse brain cells.

The scRNA-seq dataset consisting of 1.3 million cells from the cortex, hippocampus, and ventricular zones of embryonic day 18 mouse brains were downloaded from the 10X Genomics website and processed using the normalization and filtering steps of Zheng et al.,[14] as implemented by the python package scanpy.[15] Scanpy was also used to compute a neighborhood graph of the observations using a Gaussian kernel with adaptive widths, and then the points were clustered using the Louvain method. Subsequent analysis of this dataset was then performed in R. FIt-SNE of all 1,306,127 cells was computed with 4,000 iterations of gradient descent (2,000 of them being early exaggeration iterations) and other parameters set to defaults. FIt-SNE with the same parameters was also run on a random subset of 50,000 cells. We sought to identify known cell types from the Allen Brain Atlas (http://celltypes.brain-map.org/rnaseq/mouse) in the embedding, and gave two examples of cell populations (see Supplementary Table 9 of Tasic et al. (2018)[3]) that could be identified in the full dataset, but not in the downsampled embedding.

8.6. t-SNE heatmap of retinal cells.

The scRNA-seq retinal cells data of Shekhar et al. (2016)[11] was downloaded from GEO (GSE81905). The digital expression matrix was preprocessed using the code provided by the authors of the original publication (https://github.com/broadinstitute/BipolarCell2016). In short, libraries containing more than 10% mitochondrially derived transcripts were removed, cells with ≤ 500 genes were removed, as were genes with expression in ≤ 30 cells or having ≥ 60 transcripts, resulting in 13,166 genes and 27,499 cells. Finally, the data were median normalized, log-transformed, and the genes were Z-scored. The top 37 principal components were computed and used as input to 1D FIt-SNE with perplexity 30 and for 1000 iterations. Finally, the t-SNE heatmap (Figure 2) was computed as described in the main text, with the marker genes (Tacr3, Rcvrn, Syt2, Irx5, Irx6, Vsx1, Hcn4, Grik1, Gria1, Kcng4, Hcn1, Cabp5, Grm6, Isl1, Scgn, Otx2, Vsx2, Car8, Sebox, Prkca) from Shekhar et al. (2016)[11] listed in Supplemental Table 2. Each marker gene was enriched with the 25 genes with most similar expression patterns. Genes associated with each cluster in the 2D embedding were obtained by running dbscan on the 2D t-SNE with the settings $\epsilon = 2$ and a minimum number of points of 40. For each cluster i, a "metagene" c_i of length 27,499 was generated, where $c_i(k) = 1$ if the kth cell is in the ith cluster and $c_i(k) = 0$ otherwise. These vectors were then treated as "genes" and enriched in the same fashion as the genes.

8.7. Comparing approximate nearest neighbors and VP trees on scRNA-seq data.

To evaluate the effect of approximate nearest neighbors on embedding quality of scRNA-seq data, we compared the resulting embeddings on several scRNA-seq datasets where labels are predetermined by other sources. For each dataset, we also compute the 1-nearest neighbor error (1N error), defined as the percentage of cells for which the cell closest to them in the embedding belongs to a different label. We did the comparison on the 1.3 million mouse brain cells from above, purified PBMC populations from Zheng et al. (2017),[14] and mouse visual cortex cells from Hrvatin et al. (2018).[16]

Filtered expression matrices for FACS purified peripheral blood monocyte (PBMC) populations were downloaded from the 10X website[14] and concatenated them to a single expression matrix. The matrix was filtered to include cells expressing more than 400 genes and gene expressed in more than 100 cells, resulting in a matrix with 83,992 cells and 12,776 genes. Purified CD4 helper T cells and cytotoxic T cells were removed, as they (by definition) are supersets of some of the other subtypes, leaving 64,664 cells. After library and log normalization, the top 25 principal components (PCs) were computed using randomized SVD.[24] FIt-SNE using VP trees and approximate nearest neighbors were was computed on the the PCs and qualitatively compared in Figure S4.

The scRNA-seq expression matrix of mouse visual cortex cells from Hrvatin et al.[16] was obtained from GEO (GSE102827). Genes with mean expression less than 0.00003 and non-zero expression in less than 4 cells were excluded, resulting in a matrix with 65,539 cells and 19,155 genes. The cells were further subsetted to those assigned to subtypes, resulting in 48,266 cells. After library and log normalization, the top 25 principal components were computed using randomized SVD. FIt-SNE using VP trees and approximate nearest neighbors were then computed on the PCs and compared in Figure S5.

9. Code Availability

FIt-SNE is available at https://github.com/KlugerLab/FIt-SNE. The code for all experiments is available at request and will be publicly available at https://github.com/KlugerLab/FIt-SNE-paper on publication.

10. Data Availability

The 1.3 million mouse brain cells dataset and FACS purified PBMCs of Zheng et al.[14] can be downloaded from 10X Genomics website (https://support.10xgenomics.com/single-cell-gene-expression/datasets/). Two other public scRNA-seq datasets from NCBI Gene Expression Omnibus (GEO) were used: Hrvatin et al. (GSE102827) and Shekhar et al. (GSE81905).

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgements

The authors would like to thank Vladimir Rokhlin, Dmitry Kobak, Mark Tygert and Jun Zhao for many useful discussions. The authors also thank Josef Spidlen and Ian Taylor for help with testing FIt-SNE on their CyTOF and scRNA-seq datasets.

GCL was supported in part by NIH grants #F30HG010102, #1R01HG008383-01A1 and U.S. NIH MSTP Training Grant T32GM007205, MR was supported in part by AFOSR grant # FA9550-16-10175 and NIH grant #1R01HG008383-01A1, SS was supported in part by the NSF (DMS-1763179) and the Alfred P. Sloan Foundation, and YK was supported in part by NIH grant #1R01HG008383-01A1.

References

[1]. Svensson Valentine, Vento-Tormo Roser, and Teichmann Sarah A. Exponential scaling of single-cell rna-seq in the past decade. Nature protocols, 13(4):599, 2018. [PubMed: 29494575]

[2]. 10X Genomics. Transciptional profiling of 1.3 million brain cells with the chromium single cell 3' solution. Application Note, 2016.

[3]. Tasic Bosiljka, Yao Zizhen, Graybuck Lucas T, Smith Kimberly A, Nguyen Thuc Nghi, Bertagnolli Darren, Goldy Jeff, Garren Emma, Economo Michael N, Viswanathan Sarada, et al. Shared and distinct transcriptomic cell types across neocortical areas. Nature, 563(7729):72, 2018. [PubMed: 30382198]

[4]. van der Maaten Laurens. Accelerating t-SNE using tree-based algorithms. Journal of machine learning research, 15(1):3221–3245, 2014.

[5]. Yianilos Peter N. Data structures and algorithms for nearest neighbor search in general metric spaces. In SODA, volume 93, pages 311–321, 1993.

[6]. Bernhardsson Erik. Annoy: Approximate nearest neighbors in c++/python optimized for memory usage and loading/saving to disk. https://github.com/spotify/annoy, 2017.

[7]. Linderman George C and Steinerberger Stefan. Clustering with t-SNE, provably. arXiv preprint arXiv:1706.02582, 2017.

[8]. Belkina Anna C, Ciccolella Christopher O, Anno Rina, Spidlen Josef, Halpert Richard, and Snyder-Cappione Jennifer. Automated optimal parameters for t-distributed stochastic neighbor embedding improve visualization and allow analysis of large datasets. bioRxiv, page 451690, 2018.

[9]. Kobak Dmitry and Berens Philipp. The art of using t-sne for single-cell transcriptomics. bioRxiv, page 453449, 2018.

[10]. Cheng Yang, Wong Michael T, van der Maaten Laurens, and Newell Evan W. Categorical analysis of human t cell heterogeneity with one-dimensional soli-expression by nonlinear stochastic embedding. The Journal of Immunology, page 1501928, 2015.

[11]. Shekhar Karthik, Lapan Sylvain W, Whitney Irene E, Tran Nicholas M, Macosko Evan Z, Kowalczyk Monika, Adiconis Xian, Levin Joshua Z, Nemesh James, Goldman Melissa, et al. Comprehensive classification of retinal bipolar neurons by single-cell transcriptomics. Cell, 166(5):1308–1323, 2016. [PubMed: 27565351]

[12]. Galili Tal, O'Callaghan Alan, Sidi Jonathan, Sievert, and Carson. heatmaply: an r package for creating interactive cluster heatmaps for online publishing. Bioinformatics, 2017.

[13]. van der Maaten Laurens and Hinton Geoffrey. Visualizing data using t-SNE. Journal of Machine Learning Research, 9(11):2579–2605, 2008.

[14]. Zheng Grace XY, Terry Jessica M, Belgrader Phillip, Ryvkin Paul, Bent Zachary W, Wilson Ryan, Ziraldo Solongo B, Wheeler Tobias D, McDermott Geoff P, Zhu Junjie, et al. Massively parallel digital transcriptional profiling of single cells. Nature communications, 8:14049, 2017.

[15]. Wolf F Alexander, Angerer Philipp, and Theis Fabian J. Scanpy: large-scale single-cell gene expression data analysis. Genome biology, 19(1):15, 2018. [PubMed: 29409532]

[16]. Hrvatin Sinisa, Hochbaum Daniel R, Nagy M Aurel, Cicconet Marcelo, Robertson Keiramarie, Cheadle Lucas, Zilionis Rapolas, Ratner Alex, Borges-Monroy Rebeca, Klein Allon M, et al. Single-cell analysis of experience-dependent transcriptomic states in the mouse visual cortex. Nature neuroscience, 21(1):120, 2018. [PubMed: 29230054]

[17]. Barnes Josh and Hut Piet. A hierarchical O(N log N) force-calculation algorithm. Nature, 324(6096):446–449, 1986.

[18]. Dahlquist Germund and Björck Åke. Numerical methods in scientific computing, volume i. Society for Industrial and Applied Mathematics, 8, 2008.

[19]. Trefethen Lloyd N. Approximation theory and approximation practice. Siam, 2013.

[20]. Abramowitz Milton and Stegun Irene A. Handbook of mathematical function: with formulas, graphs and mathematical tables In Handbook of mathematical function: with formulas, graphs and mathematical tables. Dover Publications, 1965.

[21]. Trefethen Lloyd N and Weideman JAC. Two results on polynomial interpolation in equally spaced points. Journal of Approximation Theory, 65(3):247–260, 1991.

[22]. Halko Nathan, Martinsson Per-Gunnar, Shkolnisky Yoel, and Tygert Mark. An algorithm for the principal component analysis of large data sets. SIAM Journal on Scientific computing, 33(5): 2580–2594, 2011.

[23]. Li Huamin, Linderman George C, Szlam Arthur, Stanton Kelly P, Kluger Yuval, and Tygert Mark. Algorithm 971: an implementation of a randomized algorithm for principal component analysis. ACM Transactions on Mathematical Software (TOMS), 43(3):28, 2017.

[24]. Erichson N Benjamin, Voronin Sergey, Brunton Steven L, and Kutz J Nathan. Randomized matrix decompositions using r. arXiv preprint arXiv:1608.021J8, 2016.

[25]. Halko Nathan, Martinsson Per-Gunnar, and Tropp Joel A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review, 53(2):217–288, 2011.

[26]. Witten Rafi and Candes Emmanuel. Randomized algorithms for low-rank matrix factorizations: sharp performance bounds. Algorithmica, 72(1):264–281, 2015.

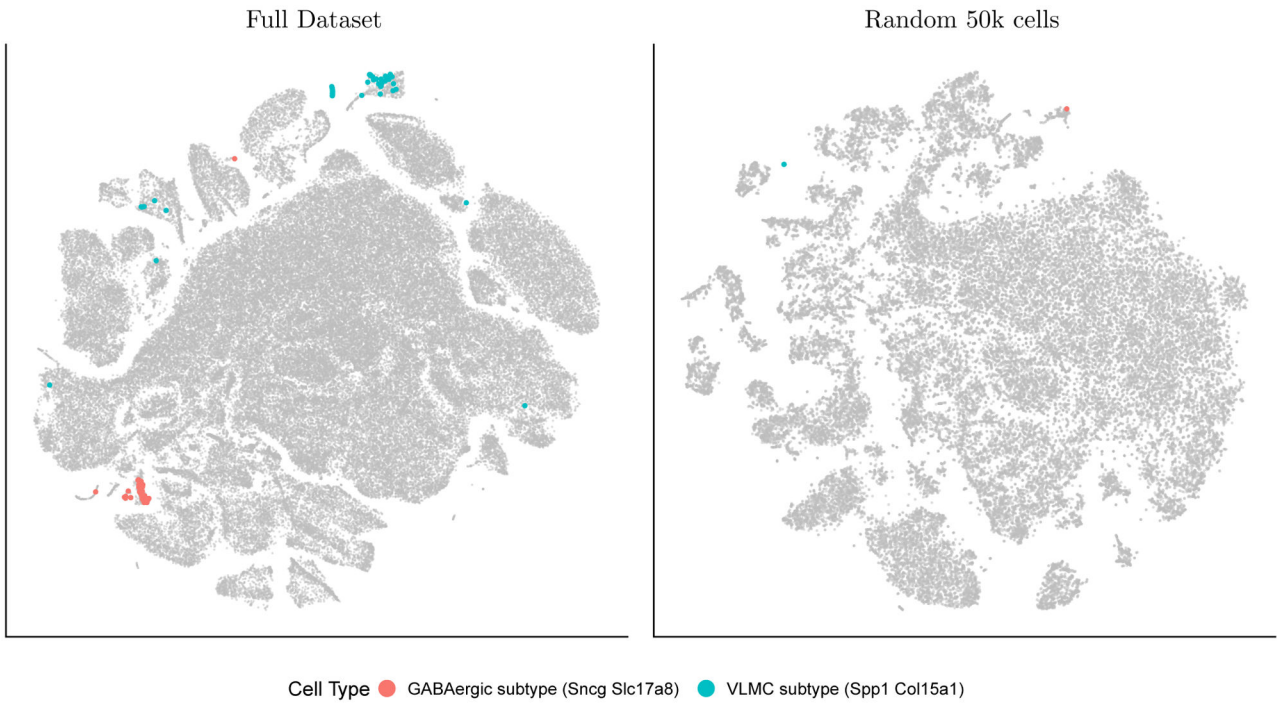

Figure 1.

FIt-SNE allows for embedding of the full 1.3 million mouse brain cell dataset (left), enabling the identification of known cell populations that cannot be identified when downsampling to a random 50,000 cells (right). (For the left figure, instead of plotting all 1.3 million embedded points, only 100,000 of the cells not expressing the marker genes are shown, whereas all the cells expressing the marker genes are shown.)

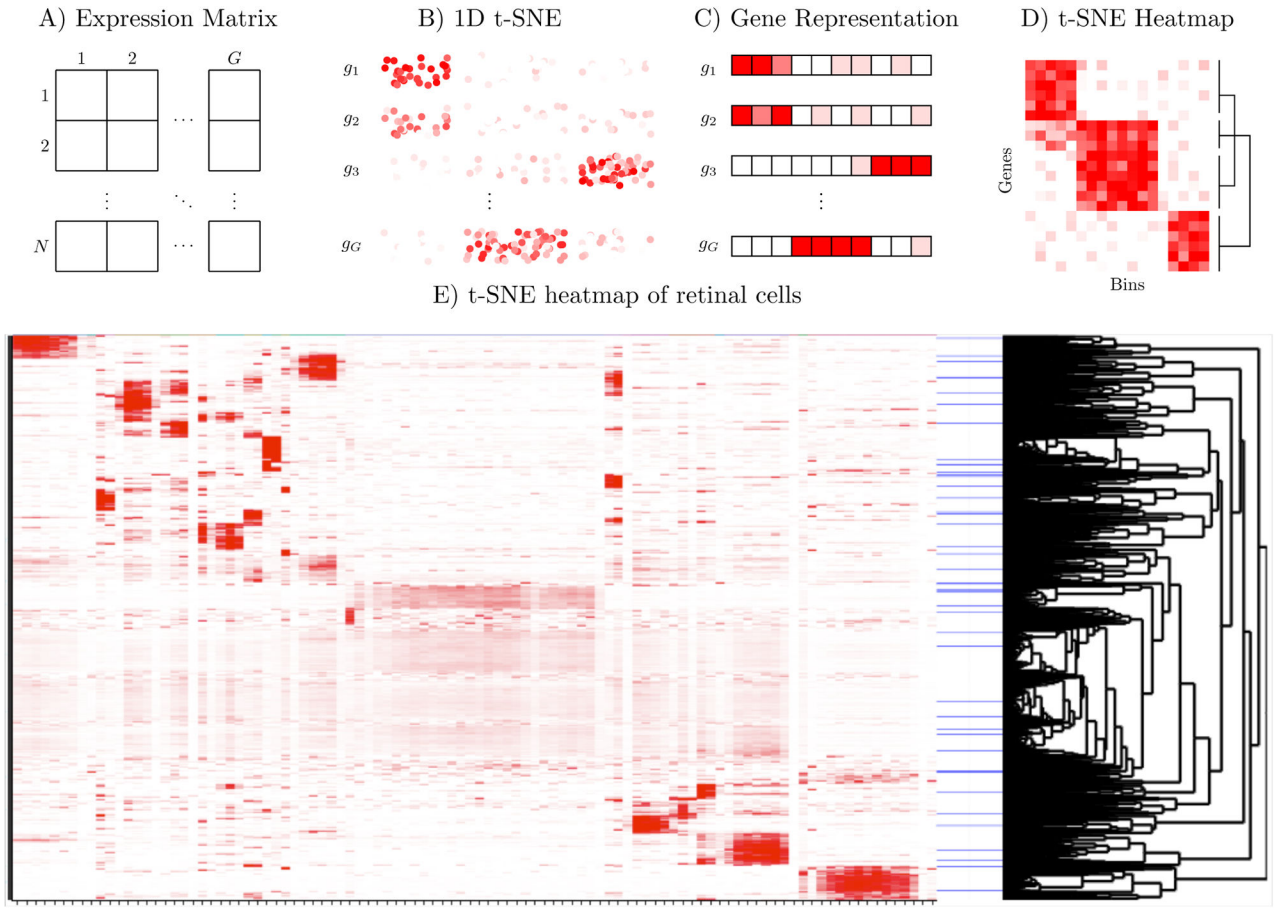

Figure 2.

Schematic and demo of t-SNE Heatmaps. Starting with the expression matrix (A) compute 1D t-SNE, which is plotted in (B) colored by the expression of each gene (with added jitter). We bin the 1D t-SNE, and represent each gene by its average expression in each bin (C), and then generate a heatmap of these vectors, so that genes with similar expression patterns in the t-SNE are grouped together (D). In (E), we demonstrate t-SNE heatmaps using retinal bipolar cells[11]

Table 1.

Time taken for 1000 iterations of the gradient descent phase of 2D t-SNE using Barnes-Hut t-SNE (BH t-SNE) compared to our implementation (FIt-SNE), as compared on a 2017 Macbook Pro for a given number of points *N*. See section 8.3.5 for more details.

N	BH t-SNE	FIt-SNE
10,000	1 min.	< 1 min.
100,000	11 min.	< 1 min.
500,000	1 hr. 10 min.	3 min.
1,000,000	3 hr. 9 min.	15 min.

Table 2.

Time taken to compute input similarities in Barnes-Hut t-SNE (vptree) compared to FIt-SNE using either multithreaded vantage-point trees (vptreeMT) or a multi-threaded approximate nearest neighbor (annMT) approach on a 2017 Macbook Pro for a given number of points *N*.

	50 Dimensions			100 Dimensions		
N	**vptree**	**vptreeMT**	**annMT**	**vptree**	**vptreeMT**	**annMT**
10,000	< 1 min.	< 1 min.	< 1 min.	< 1 min.	< 1 min.	< 1 min.
100,000	2 min.	< 1 min.	< 1 min.	3 min.	< 1 min.	< 1 min.
500,000	56 min.	15 min.	3 min.	1 hr. 30 min.	20 min.	4 min.
1,000,000	4 hr. 45 min.	1 hr. 15 min.	6 min.	7 hr. 9 min.	1 hr. 40 min.	8 min.