Published in Image Processing On Line on 2024–10–31.
Submitted on 2024–02–06, accepted on 2024–09–26.
ISSN 2105–1232 © 2024 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2024.528

A Review of t-SNE

Sangwon Jung[1], Tristan Dagobert[1], Jean-Michel Morel[2], Gabriele Facciolo[1]

[1] Université Paris-Saclay, ENS Paris-Saclay, Centre Borelli, Gif-sur-Yvette, France
{mrswjung}@gmail.com, {tristan.dagobert, gabriele.facciolo}@ens-paris-saclay.fr
[2]City University of Hong Kong, Department of Mathematics, Hong Kong
{jeamorel}@city.edu.hk

Communicated by Gregory Randall *Demo edited by* Gabriele Facciolo and Tristan Dagobert

Abstract

High dimensional data is difficult to visualize. T-Distributed Stochastic Neighbor Embedding (t-SNE) is a popular technique for dimensionality reduction enabling a planar visualization of a dataset preserving as much as possible its metric. This paper explores the theoretical background of t-SNE and its accelerated version. A comparison of the performance of t-SNE on various datasets with different dimensions is also performed.

Source Code

The source code and documentation associated to this article are available from the web page of this article[1]. The **code** uses functions from the scikit-learn Python library[2], which have **not been reviewed**. Usage instructions are included in the `README.md` file of the archive.

Keywords: dimensionality reduction; manifold learning; SNE; t-SNE; Barnes-Hut

1 Introduction

Visualization of high-dimensional data is an increasingly important problem in many domains, because understanding the data's global and local structure by using a visualization tool may lead to conceive efficient data processes adapted to these data. Dimensionality reduction techniques transform a high-dimensional dataset $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ of dimension d into a two or three dimensional dataset $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_N\}$ that can be displayed on a scatter-plot. The goal of such dimensionality reduction is to maintain as much metric information as possible from the high-dimensional data in the low-dimensional map, while enabling direct visualization of the transformed dataset. Dimension reduction techniques can be classical methods such as Principal Component Analysis [28], [17], Multi-Dimensional Scaling [4], or more modern methods such as Sammon Mapping [20], Curvilinear Component Analysis [5], Stochastic Neighbor Embedding [7], Isomap [22], Locally Linear Embedding

[1] https://doi.org/10.5201/ipol.2024.528
[2] https://scikit-learn.org/stable/

(LLE) [18] and Laplacian Eigenmaps [2]. Nevertheless the above mentioned methods are not guided by the goal of a direct two dimensional data visualization.

The t-distributed stochastic neighbor embedding (t-SNE) algorithm proposed by Van der Maaten et al. [25] fills nicely in that gap. It is based on the Stochastic Neighbor Embedding (SNE) algorithm proposed by Roweis & Hinton [7]. This dimensionality reduction technique embeds high-dimensional data for visualization in a low-dimensional space of two or three dimensions. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability.

Like SNE, t-SNE proceeds in two steps. First, the algorithm constructs a probability distribution over pairs of high-dimensional objects where similar objects are assigned a higher probability than dissimilar ones. Then, t-SNE obtains a similar probability distribution over the points in the low-dimensional map by moving them so as to minimize the Kullback-Leibler divergence D_{KL} between both distributions.

While t-SNE plots often seem to display clusters, these visual clusters can be strongly influenced by the chosen parameterization. Ghost "clusters" can be shown to form in non-clustered data [12]. To prevent this phenomenon, the parameters may have to be tuned as proposed in [16] and [26]. It is shown in [12] that t-SNE is often able to preserve initially well-separated clusters. For some parameter choices it approximates a simple form of spectral clustering.

In this paper, we describe the background and details of the t-SNE algorithm and an accelerated version [24]. The outline of the paper is as follows. In Section 2 and 3, we describe and analyze SNE and t-SNE. Section 4 presents the accelerated t-SNE [24] using a vantage point tree structure [27] and the Barnes-Hut algorithm [1]. Experimental results are shown in Section 5. Section 6 is a conclusion.

The code associated with the present paper is not a specific implementation of these methods. We take advantage of the fact that several effective numerical versions exist to date and are very popular in the scientific community[3]. They contain structures, functions or variables that have been accelerated and are, due to their intensive use, not or very little buggy. The versions used in the IPOL demonstrator correspond to those programmed in the `scikit-learn 1.2.1` library [15] by A. Fabisch, C. Moody and N. Travers. In the rest of the paper, we will specify when necessary the differences between the implementation and the general pseudo-code describing these methods in the present paper.

2 The Original t-SNE Algorithm

2.1 Basis of the t-SNE

The Stochastic Neighbor Embedding algorithm forms the basis of t-SNE. The SNE algorithm converts the high-dimensional Euclidean distances between data points of $\mathcal{X}$ into conditional probabilities that represent similarities. To this end, the similarity between two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ is represented by the relation

$$p_{j|i} = \begin{cases} \dfrac{\exp(-\left|\left|\mathbf{x}_j - \mathbf{x}_i\right|\right|^2 / (2\sigma_i^2))}{\sum_{k=1}^{N}_{k\neq i} \exp(-\left|\left|\mathbf{x}_k - \mathbf{x}_i\right|\right|^2 / (2\sigma_i^2))} & \text{if } i \neq j, \\ 0 & \text{otherwise.} \end{cases}, \tag{1}$$

[3] See https://lvdmaaten.github.io/tsne/

where σ_i^2 is the variance of a Gaussian centered at $\mathbf{x}_i$, to be specified. The pairwise distance of the low-dimensional points of $\mathcal{Y}$ is also converted into a conditional probability by

$$q_{j|i} = \begin{cases} \dfrac{\exp(-\left|\left|\mathbf{y}_j - \mathbf{y}_i\right|\right|^2)}{\sum_{\substack{k=1 \\ k\neq i}}^{N} \exp(-\left|\left|\mathbf{y}_k - \mathbf{y}_i\right|\right|^2)} & \text{if } i \neq j, \\ 0 & \text{otherwise.} \end{cases} . \tag{2}$$

The goal of the SNE algorithm is to find the low-dimensional representation that minimizes the Kullback-Leibler divergence between the distribution P^{SNE}, which is the conditional probability distribution over all the data points of $\mathcal{X}$, and the distribution Q^{SNE} which represents the conditional probability distribution over all the map points of $\mathcal{Y}$. The divergence to minimize is then expressed by

$$D_{\text{KL}}^{\text{SNE}}\left(P^{\text{SNE}} \| Q^{\text{SNE}}\right) = \sum_{i=1}^{N} \sum_{j=1}^{N} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}. \tag{3}$$

The t-SNE algorithm builds upon a symmetrized version of SNE, which also minimizes the Kullback-Leibler divergence between the joint probability distributions P and Q defined in the high- and low-dimensional spaces, respectively.

2.2 High-dimensional Data Representation

At first glance, a reasonable way to define the joint probability p_{ij} in a symmetric SNE, would be to set

$$p_{ij} = \frac{\exp(-\left|\left|\mathbf{x}_j - \mathbf{x}_i\right|\right|^2 / (2\sigma_i^2))}{\sum_{k=1}^{N} \sum_{\substack{l=1 \\ l\neq k}}^{N} \exp(-\left|\left|\mathbf{x}_k - \mathbf{x}_l\right|\right|^2 / (2\sigma_i^2))}.$$

However, if all the pairwise distances between a point $\mathbf{x}_i$ and the other data points are large, the value of p_{ij} will be extremely small for all j (we denote by D_{ij} the squared pairwise distances matrix whose elements are $d_{ij} = \|\mathbf{x}_j - \mathbf{x}_i\|^2$). In that case, the drawback is that the low dimensional point $\mathbf{y}_j$ will have little effect on the cost function (3). Thus, the high dimensional relationship will not be well represented. To overcome this difficulty, Van der Maaten & Hinton [25] impose to the joint probability to be a symmetrized conditional probability defined by

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}, \tag{4}$$

where N is the number of data samples and $p_{j|i}$ follows (1) so that $\sum_j p_{j|i} = 1$ for all i.

The formulation (4) has two advantages. First, the joint probability distribution is well balanced, each data point $\mathbf{x}_i$ having a similar influence in the D_{KL} minimization whose formulation is now

$$D_{\text{KL}}(P||Q) = \sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{5}$$

As Van der Maaten & Hinton [25] explain: "The similarity of datapoint $\mathbf{x}_j$ to datapoint $\mathbf{x}_i$ is the conditional probability, $p_{j|i}$, that $\mathbf{x}_i$ would pick $\mathbf{x}_j$ as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at $\mathbf{x}_i$". Second, from (4) one obtains that $p_{ij} = p_{ji}$, $p_{ii} = 0$ and $\sum_{i,j} p_{ij} = 1$, which ensures that $\sum_j p_{ij} > \frac{1}{2N}$ for all data points $\mathbf{x}_i$. As a result each $\mathbf{x}_i$ has a significant effect on D_{KL}.

However, since the Gaussian kernel uses the Euclidean distance $\|\mathbf{x}_i - \mathbf{x}_j\|$, it is affected by the curse of dimensionality: in high dimensional data when distances lose the ability to discriminate, the

p_{ij} may become too similar (asymptotically, they would converge to a constant). To alleviate this phenomenon, it has been proposed in [21] to adjust the distances with a power transform, based on the intrinsic dimension of each data point.

2.3 Model Perplexity

In the original SNE [7], Hinton & Roweis observe that for proper modeling, the value of σ_i should be chosen according to the density of each data point $\mathbf{x}_i$. Indeed in denser regions, a smaller value of σ_i is more appropriate than in sparser regions. The authors propose to adjust the standard deviation σ_i so that the number of effective local neighbors of $\mathbf{x_i}$, i.e. those which have a significant weight in the Kullback-Leibler divergence, be nearly the same for all i. The authors appeal to the concept of perplexity from information theory, which can be interpreted as an approximation of such an effective number of neighbors. Denoting by $P_i = \{p_{j|i}\}_{1\leq j\leq N}$ the vector of all $p_{j|i}$ for a given i, the perplexity is defined by

$$\mathrm{Perp}(P_i) = 2^{H(P_i)}, \tag{6}$$

where $H(P_i) = -\sum_j p_{j|i} \log p_{j|i}$ is the Shannon entropy of P_i. From definition (1), one can deduce that

$$H(P_i) = \sum_{j=1}^{N} \left[\frac{\exp(-\left|\left|\mathbf{x}_j - \mathbf{x}_i\right|\right|^2 / (2\sigma_i^2))}{\sum_{\substack{k=1 \\ k\neq i}}^{N} \exp(-\left|\left|\mathbf{x}_k - \mathbf{x}_i\right|\right|^2 / (2\sigma_i^2))} \left(\frac{\left|\left|\mathbf{x}_j - \mathbf{x}_i\right|\right|^2}{2\sigma_i^2} + \log \sum_{\substack{k=1 \\ k\neq i}}^{N} \exp\left(\frac{-\left|\left|\mathbf{x}_k - \mathbf{x}_i\right|\right|^2}{2\sigma_i^2} \right) \right) \right]. \tag{7}$$

The bandwidth σ_i is found by a binary search method to reach a prescribed perplexity Perp^* for each i in the course of the computation of the matrix $P_{j|i} = \{p_{j|i}\}_{1\leq j\leq N}^{1\leq i\leq N}$ of all conditional probabilities. Concerning the value of Perp^*, experiments conducted in [25] and [26], show that it depends mainly of the data quantity and can lead to divergence if it is inappropriate at-all. As a trade-off [26] and [9] suggest to apply the rule $\mathrm{Perp}^* \sim \frac{N}{100}$.

The pseudo-code is shown in Algorithm 1 and is implemented this way in the `scikit-learn` library (module `manifold/_utils.pyx`).

2.4 Low-dimensional Data Representation

The t-SNE algorithm aims at learning a low-dimensional map $\mathcal{Y}$ (of dimension typically chosen as 2 or 3) that reflects the similarities p_{ij} as well as possible. To this end, it measures similarities q_{ij} between two map points $\mathbf{y}_i$ and $\mathbf{y}_j$, using an approach similar to SNE. The expression of q_{ij} could use a Gaussian formulation like that of (2). However, according to Van der Maaten & Hinton [25], the use of Gaussians in the low-dimensional space makes it difficult to optimize the cost function (5), because the so-called "crowding problem" occurs: clusters of points tend to form in the low dimensional space due to the excessively fast decay of long-range repulsive forces. To overcome this problem, t-SNE uses a heavy-tailed Student t-distribution with one degree of freedom to compute the similarities of the low-dimensional map points. This enables dissimilar objects to keep interacting. The joint probability therefore is defined by

$$q_{ij} = \begin{cases} \dfrac{(1 + \left|\left|\mathbf{y}_i - \mathbf{y}_j\right|\right|^2)^{-1}}{\sum_{k=1}^{N} \sum_{\substack{l=1 \\ l\neq k}}^{N} (1 + \left|\left|\mathbf{y}_k - \mathbf{y}_l\right|\right|^2)^{-1}} & \text{if } i \neq j, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

In the following we denote by Q_{ij} the matrix whose elements are q_{ij} for $1 \leq i \leq N$ and $1 \leq j \leq N$.

Algorithm 1: Conditional probabilities computation with a given model perplexity.

Input D_{ij}**:** Pairwise distances matrix between the datapoints of $\mathcal{X}$.
Input K**:** Maximal number of iterations.
Input Perp^***:** Optimal perplexity to reach.
Output $P_{j|i}$**:** Matrix of the conditional probabilities i.e. $\{p_{j|i}\}_{1\leq j\leq N}^{1\leq i\leq N}$.

$H^* \leftarrow \log \mathrm{Perp}^*$
for $i = 1, \ldots, N$ **do**
  $\beta_{\min} \leftarrow -\infty$
  $\beta_{\max} \leftarrow +\infty$
  $\beta \leftarrow 1$
  $\Delta H \leftarrow +\infty$
  $k \leftarrow 1$
  while $\Delta H \geq \epsilon$ *and* $k \leq K$ **do**
    $\sigma_i \leftarrow \sqrt{2\beta}$
    // Computation of the conditional probabilities
    for $j = 1, \ldots, N$ **do**
      $p_{j|i} \leftarrow$ according to (1) $[D_{ij}, \sigma_i]$
    // Computation of the entropy
    $H(P_i) \leftarrow$ according to (7) $[P_i, D_{ij}, \sigma_i]$
    $\Delta H \leftarrow H(P_i) - H^*$
    // Adaptation of β
    if $\Delta H > 0$ **then**
      $\beta_{\min} \leftarrow \beta$
      $\beta \leftarrow 2\beta$ **if** $\beta_{\max} = +\infty$ **else** $(\beta + \beta_{\max})/2$
    else
      $\beta_{\max} \leftarrow \beta$
      $\beta \leftarrow \beta/2$ **if** $\beta_{\min} = -\infty$ **else** $\beta \leftarrow (\beta + \beta_{\min})/2$
  return $P_{j|i}$

2.5 Minimization of the Kullback-Leibler Divergence

The locations of the points $\mathbf{y}_i$ in the map are determined by minimizing the Kullback-Leibler divergence (5) between the distributions P and Q with respect to the points $\mathbf{y}_i$. This minimization is performed by the descent of the gradient whose formulation, proved in the paper [25, Appendix A], is

$$\frac{\partial D_{\mathrm{KL}}}{\partial \mathbf{y}_i} = 4 \sum_{j=1}^{N} (p_{ij} - q_{ij})(\mathbf{y}_i - \mathbf{y}_j)(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2)^{-1}. \tag{9}$$

The gradient descent with time step t follows a two-step scheme involving $\mathbf{y}_i^{(t)}$ and $\mathbf{y}_i^{(t-1)}$ and is composed of an adaptive learning rate $\eta_i(t) > 0$ and an adaptive momentum $\alpha(t) \in [0; 1]$ based on the Jacobs strategy [8] to increase the convergence rate. Its formulation for all i is

$$\mathbf{y}_i^{(t+1)} = \mathbf{y}_i^{(t)} - \eta_i(t) \frac{\partial D_{\mathrm{KL}}}{\partial \mathbf{y}_i}(t) + \alpha(t-1)(\mathbf{y}_i^{(t)} - \mathbf{y}_i^{(t-1)}). \tag{10}$$

Adaptive term $\eta_i(t)$. The learning rate $\eta_i(t)$ is adapted according to the so-called "delta-bar-delta" Jacobs method [8, eq. (4)],

$$\eta_i(t+1) = \begin{cases} \eta(t) + \kappa & \text{if } \bar{\delta}(t-1)\delta(t) > 0, \\ \eta(t)(1-\phi) & \text{if } \bar{\delta}(t-1)\delta(t) < 0, , \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

where $\delta(t) = \frac{\partial D_{\mathrm{KL}}}{\partial \mathbf{y}_i}(t)$, $\bar{\delta}(t) = (1-\theta)\delta(t) + \theta\bar{\delta}(t-1)$ with parameters $\kappa \in \mathbb{R}, \phi \in [0;1]$ and $\theta \in [0;1]$ arbitrary fixed. The implementation in `scikit-learn` differs from (11) with the formulation

$$\eta_i(t+1) = \begin{cases} \eta(t) + 0.2 & \text{if } (\eta_i(t) - \eta_i(t-1))\delta(t) < 0, \\ \eta(t) \times 0.8 & \text{otherwise.} \end{cases} \tag{12}$$

These differences are not explained. Our interpretation is that if θ is set to 0 and if the gradient descent scheme is virtually simplified as $\mathbf{y}_i^{(t+1)} = \mathbf{y}_i^{(t)} - \eta_i(t)\frac{\partial D_{\mathrm{KL}}}{\partial \mathbf{y}_i}(t)$ then $\mathrm{sign}(\eta_i(t) - \eta_i(t-1)) = -\mathrm{sign}(\delta(t-1))$ so that the inequality tests in (11) and (12) are equivalent.

Adaptive term $\alpha(t)$. The momentum term $\alpha(t)$ was first introduced by Jacobs in order to accelerate the convergence of the gradient from the formulation [8, eq. (2)]

$$\mathbf{y}_i^{(t+1)} = \mathbf{y}_i^{(t)} - (1-\alpha(t))\eta\frac{\partial D_{\mathrm{KL}}}{\partial \mathbf{y}_i}(t) + \alpha(t-1)(\mathbf{y}_i^{(t)} - \mathbf{y}_i^{(t-1)}). \tag{13}$$

According to Jacobs [8] "When consecutive derivatives of a weight (i.e. $\mathbf{y_i}^t$) possess the same sign, the exponentially weighted sum grows large in magnitude and the weight is adjusted by a large amount. Similarly, when consecutive derivatives of a weight possess opposite signs, this sum becomes small in magnitude and the weight is adjusted by a small amount". Both Van der Maaten [25] and the `scikit-learn` implementation keep only the right term $\alpha(t-1)(\mathbf{y}_i^{(t)} - \mathbf{y}_i^{(t-1)})$ of (13) into (10). In the `scikit-learn` library (module `manifold/_t_sne.py`), $\alpha(t) = 0.5$ for $0 \leq t \leq 250$ and $\alpha(t) = 0.8$ for $t > 250$.

The overall pseudo-code of the t-SNE algorithm is given in Algorithm 2. It follows the implementation of `scikit-learn`.

3 The Accelerated Tree-based t-SNE Algorithm

The pristine t-SNE algorithm is computationally expensive. Indeed, to compute the high-dimensional data representation, we need to compute the pairwise distance between all of the high-dimensional data points. For N points in k dimensions the cost of this step is $O(N^2k)$. In addition to that, to compute the gradient of t-SNE, we have to perform additional $O(N^2)$ operations.

Several techniques have been presented to reduce this computational complexity. In [23] Van der Maaten proposes to accelerate the gradient computation using the graph Laplacian of the matrices P_{ij} and Q_{ij}. In [24] the same author proposes a more sophisticated approach using tree-based algorithms for approximating the t-SNE steps. First the input similarities are approximated by a vantage-point tree [27] built on the input data $\mathcal{X}$. Second the t-SNE gradient is approximated by the Barnes-Hut algorithm [1]. This last modification allows one to approximate the component of the gradient due to distant data points, which is the sum of the *repulsive forces*.

Algorithm 2: The original t-SNE method as implemented in `scikit-learn`.

Input $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$**:** High-dimensional data set.
Input Perp^***:** Optimal perplexity to reach.
Input K**:** Maximal number of iterations in perplexity.
Input T**:** Number of iterations.
Input $\eta_i(0)$**:** Initial learning rate.
Input $\alpha(0)$**:** Initial momentum.
Input κ, ϕ, θ**:** Parameters for the adaptive term.
Output $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_N\}$**:** Low-dimensional data representation
// Computation of the high-dimensional data representation
for $i = 1, \dots, N$; $j = 1, \dots, N$ **do**
  $D_{ij} \leftarrow \|\mathbf{x_j} - \mathbf{x_i}\|^2$
$P_{j|i} \leftarrow$ according to Algorithm 1$[D_{ij}, K, \mathrm{Perp}^*]$
$P_{ij} = (P_{j|i} + P_{j|i}^\top)/(2N)$
// Sample initial solution with a Gaussian noise
$\mathcal{Y}^{(0)} \leftarrow \mathcal{N}(0, 10^{-4}I)$
// Gradient descent over T iterations
for $t = 1, \dots, T$ **do**
  // Computation of the low-dimensional data representation
  $Q_{ij} \leftarrow$ according to (8)$[\mathcal{Y}^{(t)}]$
  // An iteration of the gradient descent
  for $i = 1, \dots, N$ **do**
    $\frac{\partial D_{\mathrm{KL}}}{\partial \mathbf{y}_i}(t) \leftarrow$ according to (9)$[P_{ij}, Q_{ij}, \mathcal{Y}^{(t)}]$
    $\eta_i(t) \leftarrow$ according to (12)$[\kappa, \phi, \theta]$
    $\mathbf{y}_i^{(t+1)} \leftarrow$ according to (10)$[\frac{\partial D_{\mathrm{KL}}}{\partial \mathbf{y}_i}(t), \eta_i(t), \mathcal{Y}^{(t)}]$
return $\mathcal{Y}^{(T+1)}$

3.1 Input Similarity Approximation Restriction to the Neighborhood

In the original approach [25], the conditional probability (1) is calculated from a normalized Gaussian distribution so that one has to sum over all the datapoints to form the denominator term. Van der Maaten [24] argues that thanks to the Gaussian formulation, the probability vanishes for dissimilar input data $\mathbf{x}_i$ and $\mathbf{x}_j$. Consequently it is possible to approximate (1) by limiting the number of components of the denominator to the closest neighbors of $\mathbf{x}_i$ as

$$p_{j|i} = \begin{cases} \frac{\exp(-||\mathbf{x}_j - \mathbf{x}_i||^2/2\sigma_i^2)}{\sum_{k \in \mathcal{N}_i} \exp(-||\mathbf{x}_k - \mathbf{x}_i||^2/2\sigma_i^2)} & \text{if } j \in \mathcal{N}_i, \\ 0 & \text{otherwise.} \end{cases}, \tag{14}$$

where $\mathcal{N}_i$ represents the set of the $\lfloor 3\mathrm{Perp}(P_i) \rfloor$ nearest neighbors of $\mathbf{x}_i$ and while the symmetrized conditional probability (4) stays unchanged.

The so-called "all nearest neighbors" problem of high-dimensional data is a well studied subject and is efficiently solved thanks to appropriate data structures (see e.g. [19]) which are in particular the binary trees. In [23], the nearest neighbors are found in time $\mathcal{O}(\mathrm{Perp}(P_i) N \log N)$ by building a vantage-point tree on the input data and performing an exact nearest-neighbor search with the help of the resulting tree. The `scikit-learn` implementation differs, replacing the vantage-point tree by a ball tree structure. Both of these binary tree structures keep the same search time complexity.

A ball tree is a graph $\mathcal{T}(V, E)$ where V and E are respectively the set of nodes and edges, and each node $v \in V$ stores the centroid $\mathbf{x}_c$, the radius r of the d-dimension ball of smallest radius which contains a subset $\mathcal{X}_v$ of the datapoints $\mathcal{X}$. In addition, each node v partitions $\mathcal{X}_v$ into two disjoint sets $\mathcal{X}_v^{\text{left}}$ and $\mathcal{X}_v^{\text{right}}$ associated respectively to two children nodes v^{left} and v^{right}. We denote by $\mathcal{T}_{\mathcal{X}_v}$ the tree whose root node is v. The building of the ball tree in the `scikit-learn` implementation follows the recursive top-down method of Omohundro [13, p. 9] where the partition in two subsets is made in three steps. First, among the d dimensions, the one along which the datapoints of $\mathcal{X}_v$ are the most spread out is selected. Second, the median value of the coordinates of these datapoints along this dimension is computed. Third, the datapoints are dispatched in $\mathcal{X}_v^{\text{left}}$ and $\mathcal{X}_v^{\text{right}}$ according to this pivot value. The pseudo-code of the ball tree building is presented in Algorithm 3 (function `_binary_tree.pxi/_recursive_build()` in the `scikit-learn` implementation).

Algorithm 3: Ball tree structure building as implemented in `scikit-learn`.

Input $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$**:** High-dimensional data set with $\mathbf{x}_i = (x_{i1}, \dots, x_{id})$.
Output $\mathcal{T}_{\mathcal{X}}$**:** Ball tree structure.

$\mathcal{T}_{\mathcal{X}} \leftarrow Ball\_tree(\mathcal{X})$
return $\mathcal{T}_{\mathcal{X}}$
function $Ball\_tree(\mathcal{X}_v)$
  while $\mathcal{X}_v \neq \emptyset$ **do**
    // Computation of the d-ball properties
    $\mathbf{x}_c \leftarrow \frac{1}{|\mathcal{X}_v|} \sum_{\mathbf{x}_i \in \mathcal{X}_v} \mathbf{x}_i$
    $r \leftarrow \max_{\mathbf{x}_i \in \mathcal{X}_v} \|\mathbf{x}_i - \mathbf{x}_c\|$
    // Computation of the dimension where x is the most spread
    $s \leftarrow \arg\max_{j \in [d]} (\max_{i \in [|\mathcal{X}_v|]} x_{ij} - \min_{i \in [|\mathcal{X}_v|]} x_{ij})$
    // Computation of the pivot value
    $\lambda \leftarrow \text{median}\{x_{is} | \mathbf{x}_i \in \mathcal{X}_v\}$
    // Partition of the datapoints
    $\mathcal{X}_v^{\text{left}} \leftarrow \{\mathbf{x}_i \in \mathcal{X}_v | x_{is} < \lambda\}$
    $\mathcal{X}_v^{\text{right}} \leftarrow \{\mathbf{x}_i \in \mathcal{X}_v | x_{is} \geq \lambda\}$
    // Building of the children trees
    $\mathcal{T}_{\mathcal{X}_v^{\text{left}}} \leftarrow Ball\_tree(\mathcal{X}_v^{\text{left}})$
    $\mathcal{T}_{\mathcal{X}_v^{\text{right}}} \leftarrow Ball\_tree(\mathcal{X}_v^{\text{right}})$
  $\mathcal{T}_{\mathcal{X}_v} \leftarrow (\mathcal{X}_v, \mathbf{x}_c, r, \mathcal{T}_{\mathcal{X}_v^{\text{left}}}, \mathcal{T}_{\mathcal{X}_v^{\text{right}}})$
  return $\mathcal{T}_{\mathcal{X}_v}$

For each data point $\mathbf{x}_i$ of $\mathcal{X}$, the $K = \lfloor 3\text{Perp}(P_i) \rfloor$ nearest-neighbor search is performed on the ball tree $\mathcal{T}_{\mathcal{X}}$ thanks to a depth-first traversal search strategy (see e.g. [19, §4.2]). The algorithm visits recursively all the sub-trees $\mathcal{T}_{\mathcal{X}_v}$ except those for which it can be determined that it is impossible that they contain any of the K nearest neighbors of $\mathbf{x}_i$. Thus, as soon as the distance between the centroid node $\mathbf{x}_c$ (stored in v and associated to $\mathcal{T}_{\mathcal{X}_v}$) and $\mathbf{x}_i$ is larger than the distance between $\mathbf{x}_i$ and the farthest neighbor among the already found K neighbors, then the sub-tree $\mathcal{T}_{\mathcal{X}_v}$ is not crossed. The pseudo-code of the K nearest-neighbor search is given in Algorithm 4 (function `_binary_tree.pxi/query()` in the `scikit-learn` implementation).

Algorithm 4: Search for the K nearest neighbors sets $(\mathcal{N}_i)_{1\leq i\leq N}$ in the ball tree as implemented in `scikit-learn`.

Input $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$**:** High-dimensional data set.
Input $\mathcal{T}_\mathcal{X}$**:** Ball tree associated to $\mathcal{X}$.
Input K**:** The number of nearest neighbors to retrieve.
Output $(\mathcal{N}_i)_{1\leq i\leq N}$**:** The K nearest neighbors sets.

for $i = 1, \dots, N$ **do**
  $\mathcal{N}_i \leftarrow \emptyset$
  $\mathcal{N}_i \leftarrow KNearestNeighbors(K, \mathbf{x}_i, \mathcal{N}_i, \mathcal{T}_\mathcal{X})$
return $(\mathcal{N}_i)_{1\leq i\leq N}$
function *KNearestNeighbors*$(K, \mathbf{x}, \mathcal{N}, \mathcal{T}_\mathcal{X})$
  `// Properties of the farthest neighbor of x`
  $r_{\text{farthest}} \leftarrow \max_{\mathbf{z}\in\mathcal{N}} ||\mathbf{z} - \mathbf{x}||^2$
  $\mathbf{x}_{\text{farthest}} \leftarrow \arg\max_{\mathbf{z}\in\mathcal{N}} ||\mathbf{z} - \mathbf{x}||^2$
  `// Get the current node properties`
  $(\mathcal{X}, \mathbf{x}_c, r, \mathcal{T}_{\mathcal{X}^{\text{left}}}, \mathcal{T}_{\mathcal{X}^{\text{right}}}) \leftarrow \mathcal{T}_\mathcal{X}$
  if $||\mathbf{x} - \mathbf{x}_c||^2 \leq r_{\text{farthest}}$ **then**
    `// Case where` $\mathcal{T}_\mathcal{X}$ `is a leaf node, i.e.` $\mathcal{X}$ `is a singleton`
    if $|\mathcal{X}| = 1$ **then**
      if $|\mathcal{N}| = K \wedge r_{\text{farthest}} > ||\mathbf{z}_{\in\mathcal{X}} - \mathbf{x}||^2$ **then**
        `// We replace the farthest neighbor by a closer if needed`
        $\mathcal{N} \leftarrow \mathcal{N} \setminus \mathbf{x}_{\text{farthest}} \cup \mathcal{X}$
      else
        $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{X}$
    else
      `// General case where we are in an internal node`
      $(\mathcal{X}^{\text{left}}, \mathbf{c}^{\text{left}}, \dots) \leftarrow \mathcal{T}_{\mathcal{X}^{\text{left}}}$
      $(\mathcal{X}^{\text{right}}, \mathbf{c}^{\text{right}}, \dots) \leftarrow \mathcal{T}_{\mathcal{X}^{\text{right}}}$
      if $||\mathbf{x} - \mathbf{c}^{left}||^2 \leq ||\mathbf{x} - \mathbf{c}^{right}||^2$ **then**
        $\mathcal{N} \leftarrow KNearestNeighbors(K, \mathbf{x}, \mathcal{N}, \mathcal{T}_{\mathcal{X}^{\text{left}}}) \cup \mathcal{N}$
        $\mathcal{N} \leftarrow KNearestNeighbors(K, \mathbf{x}, \mathcal{N}, \mathcal{T}_{x^{\text{right}}}) \cup \mathcal{N}$
      else
        $\mathcal{N} \leftarrow KNearestNeighbors(K, \mathbf{x}, \mathcal{N}, \mathcal{T}_{x^{\text{right}}}) \cup \mathcal{N}$
        $\mathcal{N} \leftarrow KNearestNeighbors(K, \mathbf{x}, \mathcal{N}, \mathcal{T}_{x^{\text{left}}}) \cup \mathcal{N}$
  return $\mathcal{N}$

3.2 The Barnes-Hut Approximation of the Gradient

In [24], Van der Maaten reformulates the gradient (8) so as to present it in the form of a particle attractive-repulsive dynamical system (see e.g. [3]). Setting $Z = \sum_{k=1}^{N} \sum_{\substack{l=1 \\ l \neq k}}^{N} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}$, it follows from (8) and (9) that

$$\frac{\partial D_{\mathrm{KL}}}{\partial \mathbf{y}_i} = 4 \sum_{j=1}^{N} (p_{ij} - q_{ij}) q_{ij} Z (\mathbf{y}_i - \mathbf{y}_j), \tag{15}$$

$$= 4 \left(\underbrace{\sum_{j \neq i} p_{ij} q_{ij} Z (\mathbf{y}_i - \mathbf{y}_j)}_{F_{\mathrm{attr}}} - \underbrace{\sum_{j \neq i} q_{ij}^2 Z (\mathbf{y}_i - \mathbf{y}_j)}_{F_{\mathrm{rep}}} \right), \tag{16}$$

where F_{attr} (resp. F_{rep}) represents the sum of all attractive (resp. repulsive) forces and the terms $\mathbf{y}_i$ identify the particle positions. Van der Maaten shows [24, §4.1] that thanks to the approximation of the matrix $P_{j|i}$ made by its sparse representation (14), the computation of F_{attr} can be done efficiently with a complexity of order $\mathcal{O}(\lfloor 3\mathrm{Perp}(P_i) \rfloor N)$. However computing F_{rep} is of order $\mathcal{O}(N^2)$ which stays computationally expensive [1]. The Barnes-Hut algorithm [1] overcomes this drawback by approximating F_{rep} in $\mathcal{O}(N \log N)$ thanks to a quad-tree representation followed by a depth-first search [24].

A quad-tree is a class of graph that we denote by $\mathcal{Q}(\mathcal{V}, \mathcal{E})$, where V and E are respectively the set of nodes and edges. In our case, in a quad-tree each node $v \in V$ corresponds to a square cell in the two-dimensional space, which contains a subset $\mathcal{Y}_v$ of the map points set $\mathcal{Y}$ and stores their center of mass $\mathbf{y}_c$. In addition, each node v partitions $\mathcal{Y}_v$ into four disjoint subsets $\mathcal{Y}_v^{\mathrm{nw}}$, $\mathcal{Y}_v^{\mathrm{ne}}$, $\mathcal{Y}_v^{\mathrm{sw}}$ and $\mathcal{Y}_v^{\mathrm{se}}$ associated respectively to four children nodes v^{nw}, v^{ne}, v^{sw} and v^{se} which divide the two-dimensional space into four quadrants, northwest, northeast, southwest and southeast respectively, meeting at the center of the parent node cell. We denote by $\mathcal{Q}_{\mathcal{Y}_v}$ the quadtree whose root node is v. The idea of the approximation is that if the cell v is small and far enough from the target point $\mathbf{y}_i$, the contribution of the map points repulsion force inside the cell v can be similar to the one exerted by the center of mass $\mathbf{y}_c$ of the cell, weighted by the number of map points inside v

$$\sum_{\mathbf{y_j} \in \mathcal{Y}_v} -q_{ij}^2 Z (\mathbf{y}_i - \mathbf{y}_j) \approx -|\mathcal{Y}_v| q_{i,c}^2 Z (\mathbf{y}_i - \mathbf{y}_c). \tag{17}$$

The condition proposed by Barnes-Hut algorithm decides whether the cell can be used as a summary of all the map points inside it. This condition compares the distance between $\mathbf{y}_i$ and $\mathbf{y}_c$ and the size of the cell by checking if

$$\frac{d_v}{||\mathbf{y}_i - \mathbf{y}_c||^2} < \theta, \tag{18}$$

where d_v represents the diagonal of the cell and θ is a fixed threshold. If $\theta = 0$, all the pairwise interactions are computed independently, which boils down to the naive t-SNE. If $\theta > 0$ and the condition (18) is satisfied, then the cell can be used as a summary of the map points it contains. The implementation of the Barnes-Hut gradient term varies in the `scikit-learn` library (module `manifold/_barnes_hut_tsne.pyx`) because the computation of the attractive term F_{attr} is restricted to the nearest neighbors $\mathcal{N}_i$ of each $\mathbf{x_i}$. The overall pseudo-code for approximating the t-SNE gradient is given in Algorithm 5.

The overall pseudo-code of the accelerated t-SNE algorithm is given in Algorithm 6. It follows the implementation of `scikit-learn`.

Algorithm 5: Pseudo-code for approximating the t-SNE gradient using Barnes-Hut algorithm as implemented in `scikit-learn`.

Input θ**:** Threshold of cell acceptance.
Input $\mathcal{Y}$**:** Map point dataset.
Input $\mathcal{Q}_\mathcal{Y}$**:** Quadtree representation of $\mathcal{Y}$.
Input P**:** Conditional probability distribution over all the data points of $\mathcal{X}$.
Input Q**:** Conditional probability distribution over all the map points of $\mathcal{Y}$.
Output $\frac{\partial D_{\mathrm{KL}}}{\partial \mathcal{Y}}$**:** approximated gradient of t-SNE

function *BarnesHutGradient*$(\theta, \mathcal{Y}, \mathcal{Q}_\mathcal{Y}, P, Q)$
  for i **from** *1* **to** N **do**
    // Computation of the attractive force $F_{\texttt{attr}}$
    $F_{\mathrm{attr}} \leftarrow 0$
    for $j \in \mathcal{N}_i$ **do**
      $F_{\mathrm{attr}} \leftarrow F_{\mathrm{attr}} + p_{ij} q_{ij} Z(\mathbf{y_i} - \mathbf{y_j})$
    // Computation of the repulsive force $F_{\texttt{rep}}$
    $F_{\mathrm{rep}} \leftarrow 0$
    for $\mathcal{Q}_{\mathcal{Y}_v} \in \mathcal{Q}_\mathcal{Y}$ **do**
      if $d_v / ||\mathbf{y}_i - \mathbf{y}_c||^2 < \theta$ **then**
        $F_{\mathrm{rep}} \leftarrow F_{\mathrm{rep}} - |\mathcal{Y}_v| q_{i,c}^2 Z(\mathbf{y}_i - \mathbf{y}_c)$
      else
        $F_{\mathrm{rep}} \leftarrow F_{\mathrm{rep}} - \sum_{\mathbf{y_j} \in Y_v} q_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)$
  return $4(F_{attr} + F_{rep})$

4 Experiments

In this section, after introducing the data sets we employ, we evaluate the t-SNE algorithm performance when varying its main parameters on well-known datasets. The default values for these parameters in the t-SNE version of `scikit-learn` are the following: $\mathrm{Perp}^* = 30, T = 1000$ and $\eta_i(t=0) = \max(N/48, 50)$. The values of Perp^* and T follow those applied in [24].

Although the t-SNE method is originally dedicated to data visualization, we evaluate it on its ability to cluster the map points of same class. Many clustering quality measures exist (see e.g. [6]) that can be roughly divided into two categories: external comparison indices and internal comparison indices. The first category requires having two partitions because it measures their similarity. The second category only uses the partition obtained and distances between its points. Since the t-SNE does not directly provide clusters but we know the labels used in the datasets, we employ an index from the second category. In the following, we apply the silhouette index which is a widely used quality metric to estimate the embedding quality.

The silhouette index. We use the formula applied in the `scikit-learn` library. Given a partition on $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$, the silhouette index is defined for each point $\mathbf{y}_i$ of $\mathcal{Y}$ by

$$s_i = \frac{b - a}{\max(a, b)}, \tag{19}$$

where a is the average distance between $\mathbf{y}_i$ and all other points in the same class, and b is the average distance between $\mathbf{y}_i$ and all other points in the next nearest cluster. The silhouette index for $\mathcal{Y}$,

Algorithm 6: The accelerated t-SNE method as implemented in `scikit-learn`.

Input $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$**:** High-dimensional data set.
Input Perp^***:** Optimal perplexity to reach.
Input T**:** Number of iterations.
Input $\eta_i(0)$**:** Initial learning rate.
Input $\alpha(0)$**:** Initial momentum.
Input κ, ϕ, θ**:** Parameters for the adaptive term.
Output $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_N\}$**:** Low-dimensional data representation
// Computation of the high-dimensional data representation
// Ball tree structure building
$\mathcal{T}_\mathcal{X} \leftarrow \text{Ball\_Tree}(\mathcal{X})$
// Search for the k nearest neighbors sets
$(N_i)_{1 \leq i \leq N} \leftarrow$ according to algorithm 4$[\mathcal{T}_\mathcal{X}, \text{Perp}^*]$
$P_{j|i} \leftarrow$ according to (14)
$P_{ij} = (P_{j|i} + P_{j|i}^\top)/(2N)$
// Sample initial solution with a Gaussian noise
$\mathcal{Y}^{(0)} \leftarrow \mathcal{N}(0, 10^{-4}I)$
// Gradient descent over T iterations
for $t = 1, \dots, T$ **do**
  // Computation of the low-dimensional data representation
  $Q_{ij} \leftarrow$ according to (8)
  // Quadtree structure building
  $\mathcal{Q}_\mathcal{Y}(t) \leftarrow$ construction according to $\mathcal{Y}^{(t)}$.
  // Barnes-Hut gradient approximation
  $\frac{\partial D_{\text{KL}}}{\partial \mathcal{Y}}(t) \leftarrow \text{BarnesHutGradient}(\theta, \mathcal{Y}, \mathcal{Q}_\mathcal{Y}, P_{ij}, Q_{ij})$
  // An iteration of the gradient descent
  for $i = 1, \dots, N$ **do**
    $\frac{\partial D_{\text{KL}}}{\partial \mathbf{y}_i}(t) \leftarrow$ according to (9)$[P_{ij}, Q_{ij}, \mathcal{Y}^{(t)}]$
    $\eta_i(t) \leftarrow$ according to (12)$[\kappa, \phi, \theta]$
    $\mathbf{y}_i^{(t+1)} \leftarrow$ according to (10) $[\frac{\partial D_{\text{KL}}}{\partial \mathbf{y}_i}(t), \eta_i(t), \mathcal{Y}^{(t)}]$
return $\mathcal{Y}$

denoted as $s_\mathcal{Y}$, is the average of the coefficients s_i. Possible values of $s_\mathcal{Y}$ are in $[-1, 1]$ where the best value is 1 and the worst -1.

4.1 Data Sets

MNIST. The MNIST dataset [11] is a well known computer vision dataset consisting of gray scale images representing the ten handwritten digits. In the version recovered by `scikit-learn`, this dataset is composed of 70 000 images of size $28 \times 28 = 784$ pixels.

CIFAR-10. In the version recovered by `scikit-learn`, the CIFAR-10 dataset [10] consists in 60 000 images of size $3 \times 32 \times 32$ split into 10 classes with 6000 images each. The 60 000 images are divided into 50 000 train images and 10 000 test images. We shall apply t-SNE on the train dataset itself and on the embedded vectors of these data. To compute the embedded vectors, we trained a small convolutional network on the 50 000 train images using PyTorch [14]. This network

is mainly composed by two convolutional layers followed by three dense layers and produces 10-dimensional embedded vectors. Table 1 shows the architecture of the network. The network was trained to minimize the cross-entropy loss using mini-batches of size 4, learning rate of 0.001, a momentum term of 0.9 after 10 epochs. The network obtained an accuracy of 84% on the CIFAR-10 test dataset.

Layer	Output shape	Number of parameters
Input image	[3, 32, 32]	0
Convolutional 2D with 32 filters of size 5×5	[32, 28, 28]	2432
ReLU activation	[32, 28, 28]	0
Maxpooling over 2×2 patches	[32, 14, 14]	0
Convolutional 2D with 32 filters of size 5×5	[32, 10, 10]	25 632
ReLU activation	[32, 10, 10]	0
Maxpooling over 2×2 patches	[32, 5, 5]	0
Flatten	[800]	0
Dense layer 800×120	[120]	96 120
ReLU activation	[120]	0
Dense layer 120×64	[64]	7744
ReLU activation	[64]	0
Dense layer 64×10	[10]	650
Output embedded vector	[10]	0
Total parameters		132 578

Table 1: Successive layers constituting the neural network.

4.2 Results

4.2.1 MNIST

Figure 1 compares the naive t-SNE and the Barnes-Hut t-SNE algorithm on the MNIST dataset. The color of the points indicates the classes of the corresponding objects. We used 10 000 randomly sampled data points from the MNIST dataset (the same randomly selected data points for both t-SNE versions). Both embedding results are made through 1000 iterations with a perplexity of 40. The results show the strong performance of Barnes-Hut t-SNE compared to the naive version. In particular, the quality of separation of classes of both algorithms are very similar: $s_{\mathcal{Y}}(\text{naive}) = 0.311$, while $s_{\mathcal{Y}}(\text{Barnes}) = 0.327$. However, the naive t-SNE is 186 times slower than the Barnes-Hut t-SNE.

Figure 2 shows the results after applying the Barnes-Hut t-SNE algorithm on 17 500, 35 000, 52 500 and 70 000 MNIST data points. The embedding result on 17 500 points forms the right clusters, but all classes are not clearly separated, $s_{\mathcal{Y}} = 0.345$. In particular, classes such as 4 and 9 are interlaced. When it comes to 70 000 data points, the Barnes-Hut t-SNE can efficiently construct high-quality embeddings of the handwritten digit images, $s_{\mathcal{Y}} = 0.389$. Although no supervised information was used, all ten digit classes are clearly separated in an embedding. Figure 3 shows the results of processing the Barnes-Hut t-SNE algorithm on 10 000 data points with 250, 500, 1000 and 2000 iterations. The clusters with 2000 iterations are clearer than with 250 iterations: $s_{\mathcal{Y}}(250) = 0.250$ and $s_{\mathcal{Y}}(2000) = 0.336$. From 1000 iterations on, the embedding results are visually stationary while the spread of the embedding point clouds increases before becoming itself stationary.

Figure 4 shows the final embedding on the randomly sampled 10 000 data points of the MNIST data set for different perplexities. We can see that for perplexity 2 the local clusters dominate the scene, but that the global cluster is not well-formed at all, value of $s_{\mathcal{Y}}$ is only 0.171. When we move to perplexity 5, we can see the formation of local clusters, yet these are still not separated enough. Using perplexity 40, we recognize the distinct classes of the MNIST dataset t-SNE embedding. The

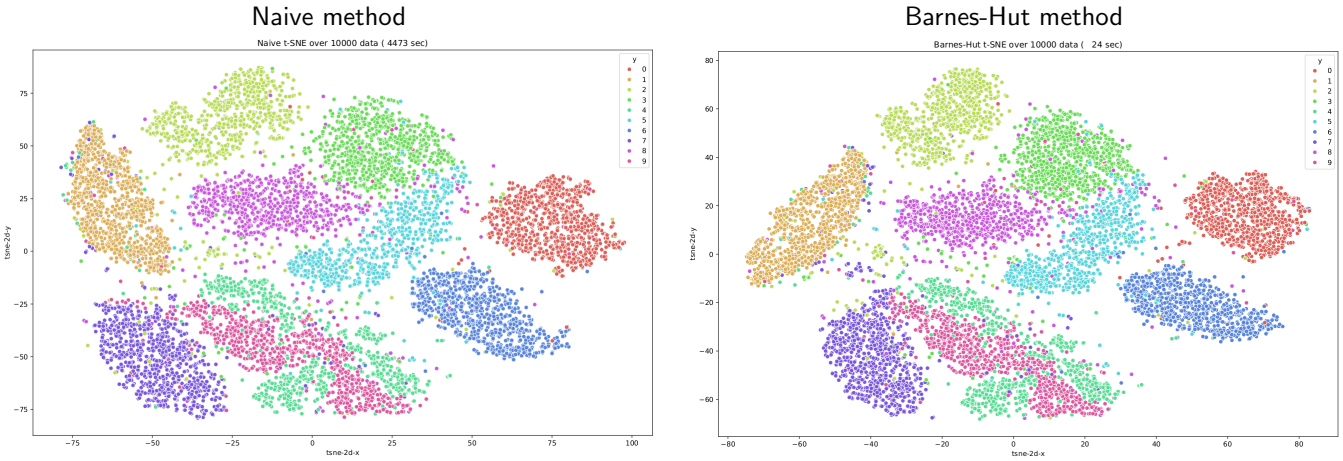


Figure 1: Naive (left column) and Barnes-Hut (right column) t-SNE visualizations obtained from 10 000 randomly picked images from the MNIST dataset. The color of the points indicates the class of the corresponding objects. Both results show similar embedding qualities, but the naive t-SNE is 186 times slower than the Barnes-Hut t-SNE.

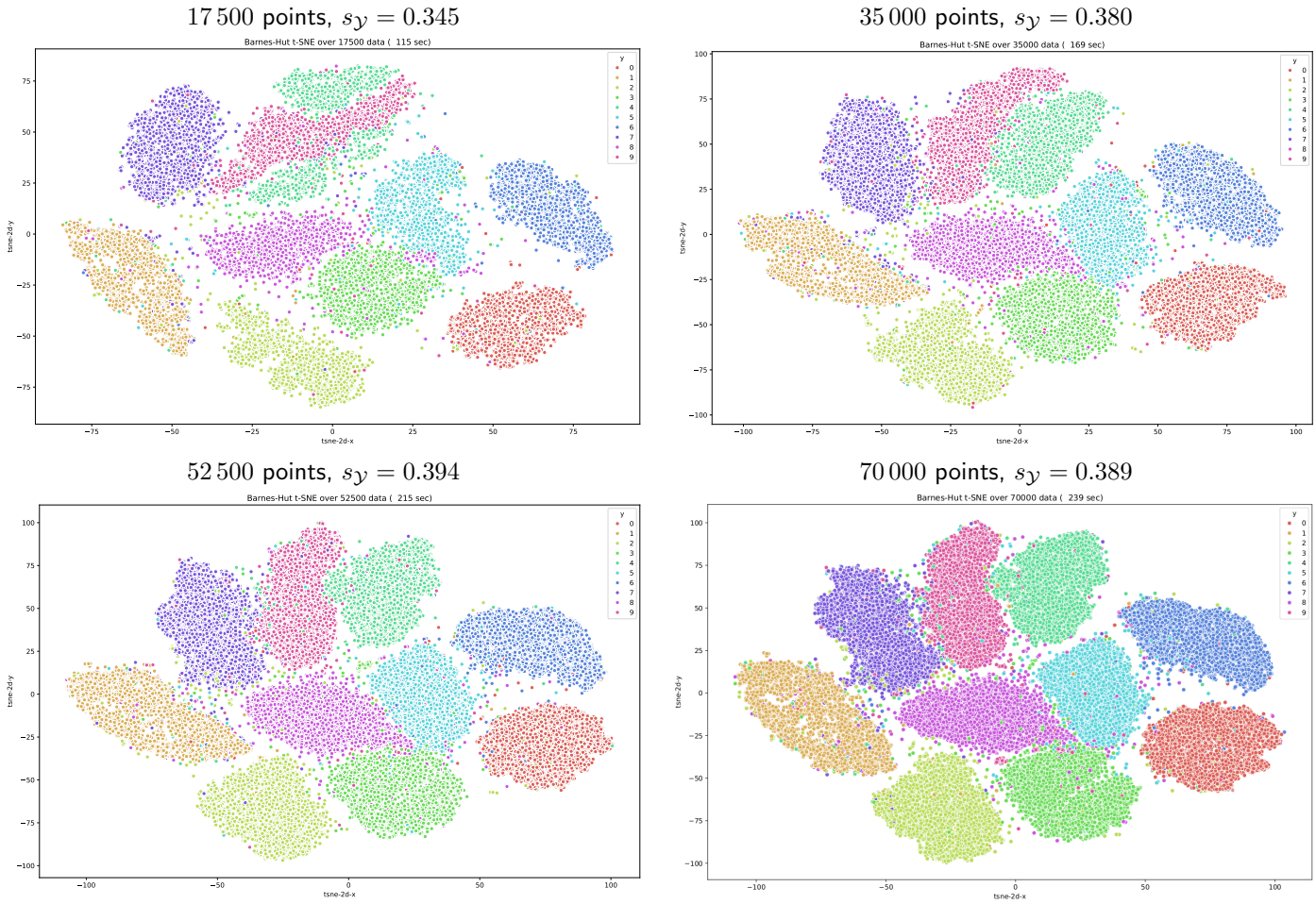


Figure 2: Results of the Barnes-Hut t-SNE algorithm on 17 500, 35 000, 52 500 and 70 000 MNIST data points. Some of the classes, such as 4 and 9, are mixed in the embedding results on 17 500 points. However, all classes are clearly separated in the embedding on 70 000 data points.

embedding result with perplexity 100, however, doesn't show much difference from the previous result.

Figure 5 shows the results of processing the Barnes-Hut t-SNE algorithm with different PCA

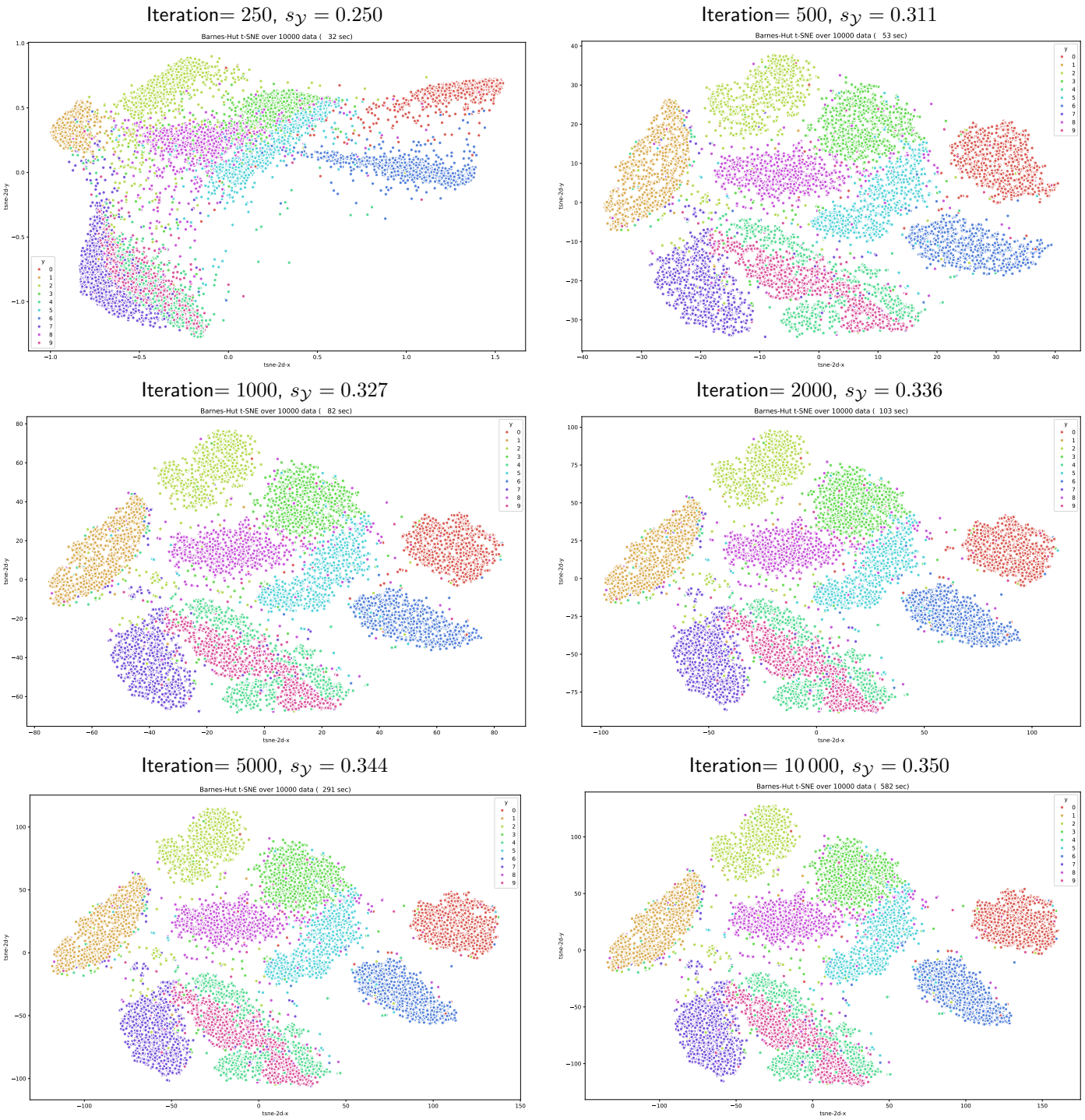


Figure 3: Results of the Barnes-Hut t-SNE algorithm on $10\,000$ data points from the MNIST dataset with respectively 250, 500, 1000,2000,5000 and $10\,000$ iterations. While 250 iterations are clearly insufficient to cluster correctly the classes, the results over 1000 iterations are similar.

dimensionality reductions (the initial dimension of the data points is 784). Using PCA with only 2 dimensions takes very little time and the points with the same classes are gathering. However, the clusters are not formed at all. When using a PCA with 10 dimensions we can see that the clusters are formed, but points from different classes are mixed. Using PCA with 50 dimensions gives results similar to an application of t-SNE without dimensionality reduction.

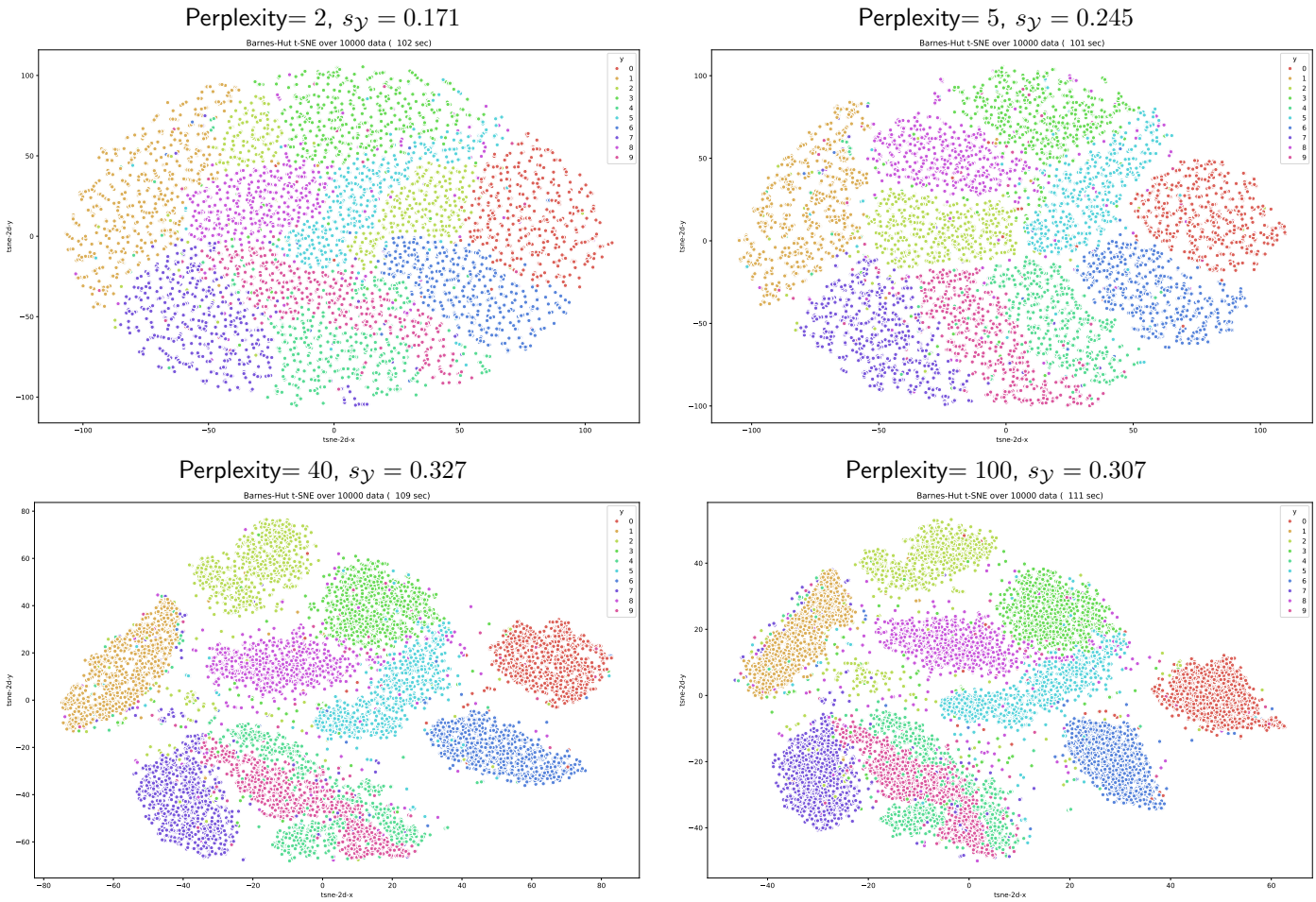


Figure 4: Results of the Barnes-Hut t-SNE algorithm on 10 000 data points from the MNIST dataset with different perplexity values. The shown plots are the embedding results with perplexities 2, 5, 40 and 100, respectively. When the perplexity is 2, the local clusters are well formed but the global cluster is not. As the perplexities increase, the class clusters get better separated.

4.2.2 CIFAR-10

Figure 6 shows the results of the Barnes-Hut t-SNE algorithm applied on the train images of the CIFAR-10 dataset and applied on their embedded vectors computed by our CNN. The color of the points indicates the classes of the corresponding images. The t-SNE algorithm used perplexity value of 40 and 1000 iterations. Unlike with the MNIST dataset, the results of t-SNE applied on the raw data show no significant cluster formation. On the contrary, the embedded vectors used as input for the t-SNE give distinct clusters.

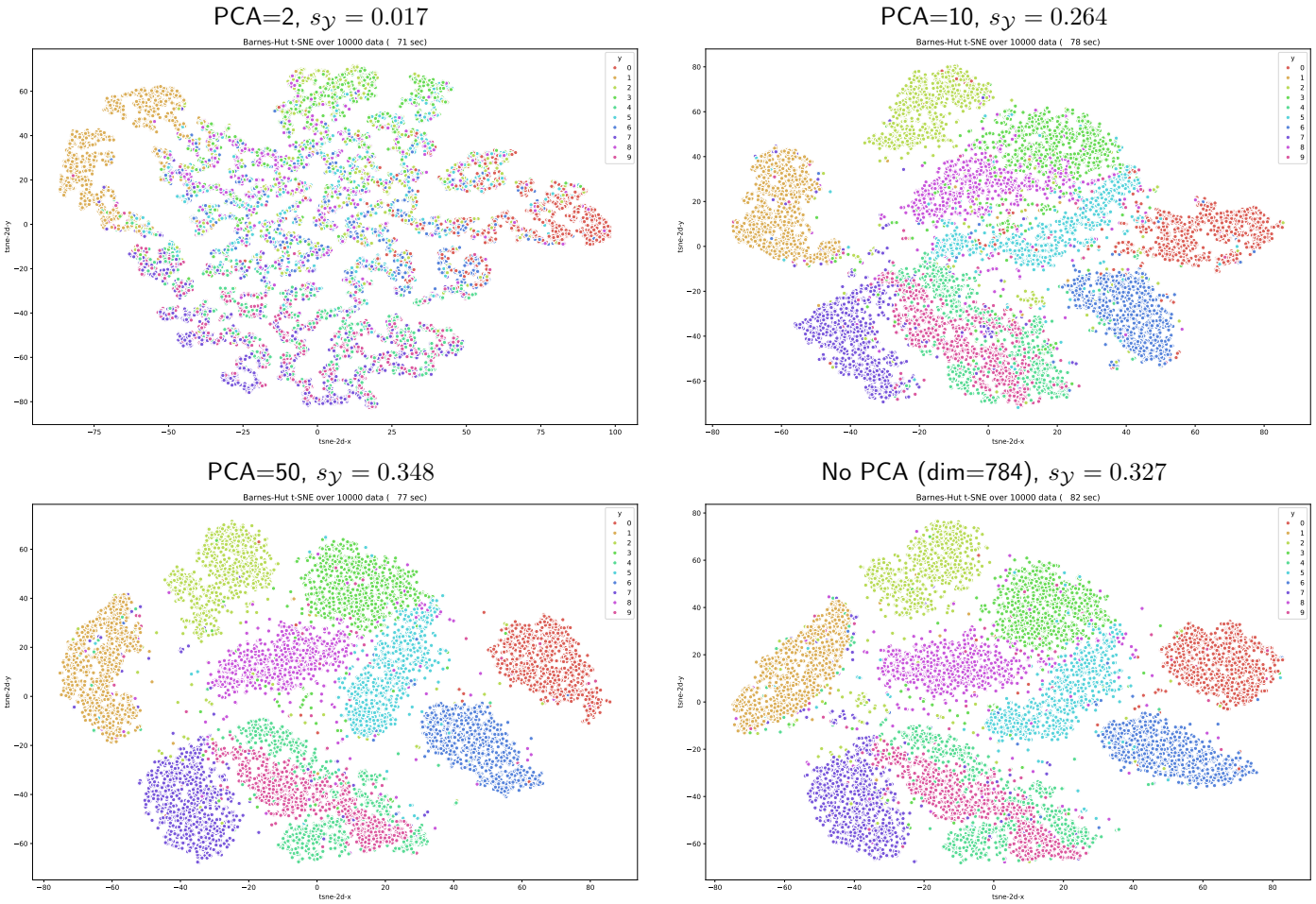
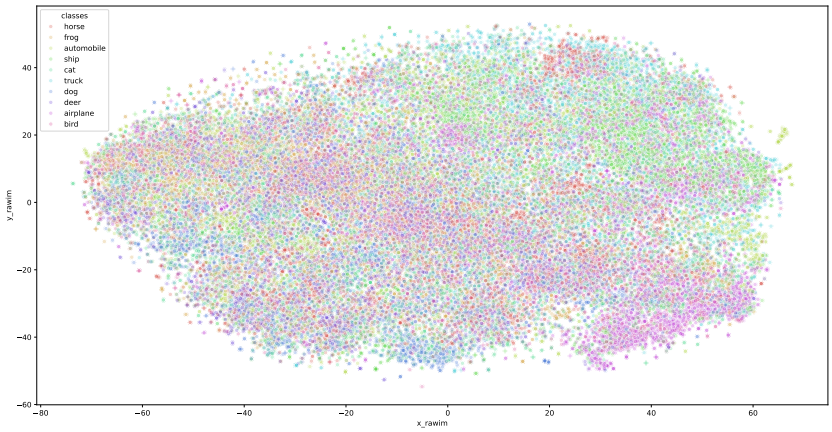


Figure 5: Results of the Barnes-Hut t-SNE algorithm on $10\,000$ data points from the MNIST dataset with different initial dimensionality reductions using PCA. While PCA=2 shows inconsistent results, using PCA with 50 dimensions gives results thar are similar to those without dimensionality reduction.

t-SNE applied on the CIFAR-10 raw data, $s_y = -0.123$



t-SNE applied on the CIFAR-10 embedded data, $s_y = 0.207$

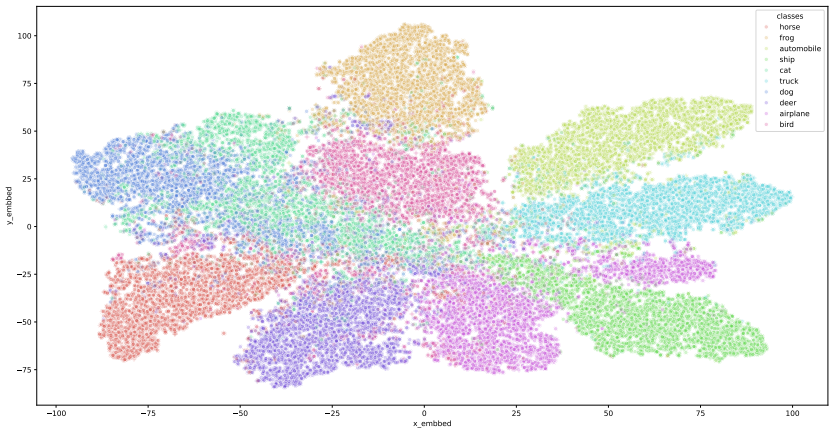


Figure 6: Results of the Barnes-Hut t-SNE on the CIFAR-10 data points. At the top, we applied the t-SNE on the CIFAR-10 data itself: no local and global structure is found. This means that t-SNE cannot capture the relations between the CIFAR-10 images by just using raw pixel values. At the bottom, the result of t-SNE applied on the output of our convolutional neural network.

5 Conclusion

In this paper, we investigated t-SNE, a dimensionality reduction algorithm for data visualization, both in its naive version and in its accelerated version by the Barnes-Hut approach. We first theoretically described the algorithms by equations and pseudo-code. One can summarize t-SNE by saying that it works by minimizing the Kullback-Leibler divergence between the distributions of the high dimensional data points and of the low dimensional map points. We also investigated how the t-SNE was accelerated. The method mainly relies on the reduction to a neighborhood of the calculus of the elementary probabilities, as well as the use of a search tree structure.

We then empirically evaluated and compared the performance of the t-SNE algorithms on different datasets. We showed that both versions of t-SNE obtain similar results but the accelerated version was clearly faster. We also studied the influence of the main parameters of the method, such as the number of iterations, the perplexity value, as well as the number of input data and the dimensionality reduction. Finally we verified how the t-SNE can be used to evaluate the performance of a neural network model by visualizing its outputs.

References

[1] J. Barnes and P. Hut, *A Hierarchical O (N Log N) Force-Calculation Algorithm*, Nature, 324 (1986), pp. 446–449, https://doi.org/10.1038/324446a0.

[2] M. Belkin and P. Niyogi, *Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering*, Advances in Neural Information Processing Systems, 14 (2001), https://doi.org/10.7551/mitpress/1120.003.0080.

[3] J. A. Carrillo, Y.-P. Choi, and S. P. Perez, *A Review on Attractive-Repulsive Hydrodynamics for Consensus in Collective Behavior*, Springer International Publishing, 2017, pp. 259–298, https://doi.org/10.1007/978-3-319-49996-3_7.

[4] M. A. Cox and T. F. Cox, *Multidimensional Scaling*, in Handbook of Data Visualization, Springer, 2008, pp. 315–347, https://doi.org/10.1007/978-3-540-33037-0_14.

[5] P. Demartines and J. Hérault, *Curvilinear Component Analysis: A Self-Organizing Neural Network for Nonlinear Mapping of Data Sets*, IEEE Transactions on Neural Networks, 8 (1997), pp. 148–154, https://doi.org/10.1109/72.554199.

[6] B. Desgraupes, *Clustering Indices*, 2016. https://api.semanticscholar.org/CorpusID:33243618.

[7] G. E. Hinton and S. Roweis, *Stochastic Neighbor Embedding*, in Advances in Neural Information Processing Systems, vol. 15, MIT Press, 2002. https://proceedings.neurips.cc/paper_files/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf.

[8] R. A. Jacobs, *Increased Rates of Convergence Through Learning Rate Adaptation*, Neural Networks, 1 (1988), pp. 295–307, https://doi.org/10.1016/0893-6080(88)90003-2.

[9] D. Kobak and P. Berens, *The Art of Using t-SNE for Single-Cell Transcriptomics*, Nature Communications, 10 (2019), https://doi.org/10.1038/s41467-019-13056-x.

[10] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, 2009. https://api.semanticscholar.org/CorpusID:18268744.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-Based Learning Applied to Document Recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324, https://doi.org/10.1109/5.726791.

[12] G. C. Linderman and S. Steinerberger, *Clustering with t-SNE, Provably*, SIAM Journal on Mathematics of Data Science, 1 (2019), pp. 313–332, https://doi.org/10.1137/18M1216134.

[13] S. M. Omohundro, *Five Balltree Construction Algorithms*, 1989. https://steveomohundro.com/wp-content/uploads/2009/03/omohundro89_five_balltree_construction_algorithms.pdf.

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Pytorch: An Imperative Style, High-Performance Deep Learning Library*, Advances in Neural Information Processing Systems, 32 (2019), https://doi.org/10.48550/arXiv.1912.01703.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-Learn: Machine Learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

[16] N. Pezzotti, B. P. Lelieveldt, L. Van Der Maaten, T. Höllt, E. Eisemann, and A. Vilanova, *Approximated and User Steerable t-SNE for Progressive Visual Analytics*, IEEE Transactions on Visualization and Computer Graphics, 23 (2016), pp. 1739–1752, https://doi.org/10.1109/TVCG.2016.2570755.

[17] V. Rokhlin, A. Szlam, and M. Tygert, *A Randomized Algorithm for Principal Component Analysis*, SIAM Journal on Matrix Analysis and Applications, 31 (2010), pp. 1100–1124, https://doi.org/10.1137/080736417.

[18] S. T. Roweis and L. K. Saul, *Nonlinear Dimensionality Reduction by Locally Linear Embedding*, Science, 290 (2000), pp. 2323–2326, https://doi.org/10.1126/science.290.5500.2323.

[19] H. Samet, *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*, Morgan Kaufmann Publishers Inc., 2005. ISBN 0123694469.

[20] J. W. Sammon, *A Nonlinear Mapping for Data Structure Analysis*, IEEE Transactions on Computers, 100 (1969), pp. 401–409, https://doi.org/10.1109/T-C.1969.222678.

[21] E. Schubert and M. Gertz, *Intrinsic t-Stochastic Neighbor Embedding for Visualization and Outlier Detection*, in International Conference on Similarity Search and Applications, Springer, 2017, pp. 188–203, https://doi.org/10.1007/978-3-319-68474-1_13.

[22] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, *A Global Geometric Framework for Nonlinear Dimensionality Reduction*, Science, 290 (2000), pp. 2319–2323, https://doi.org/10.1126/science.290.5500.2319.

[23] L. Van Der Maaten, *Fast Optimization for t-SNE*, in Neural Information Processing Systems (NIPS) Workshop on Challenges in Data Visualization, vol. 100, Citeseer, 2010. https://cseweb.ucsd.edu/~lvdmaaten/workshops/nips2010/papers/vandermaaten.pdf.

[24] ——, *Accelerating t-SNE Using Tree-Based Algorithms*, The Journal of Machine Learning Research, 15 (2014), pp. 3221–3245. https://jmlr.org/papers/volume15/vandermaaten14a/vandermaaten14a.pdf.

[25] L. Van der Maaten and G. Hinton, *Visualizing Data Using t-SNE*, Journal of Machine Learning Research, 9 (2008). https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

[26] M. Wattenberg, F. Viégas, and I. Johnson, *How to Use t-SNE Effectively*, Distill, 1 (2016), p. e2, https://doi.org/10.23915/distill.00002.

[27] P. N. Yianilos, *Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces*, in ACM-SIAM Symposium on Discrete Algorithms, vol. 66, SIAM, 1993, p. 311. https://dl.acm.org/doi/10.5555/313559.313789.

[28] H. Zou, T. Hastie, and R. Tibshirani, *Sparse Principal Component Analysis*, Journal of Computational and Graphical Statistics, 15 (2006), pp. 265–286, https://doi.org/10.1198/106186006X113430.