GPGPU Linear Complexity t-SNE Optimization

Pezzotti, Nicola; Thijssen, Julian; Mordvinstev, Alexander; Hollt, Thomas; Van Lew, Baldur; Lelieveldt, Boudewijn; Eisemann, Elmar; Vilanova , Anna

DOI
10.1109/TVCG.2019.2934307

Publication date
2020

Document Version
Accepted author manuscript

Published in
IEEE Transactions on Visualization and Computer Graphics

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2019.2934307, IEEE Transactions on Visualization and Computer Graphics

GPGPU Linear Complexity t-SNE Optimization

Nicola Pezzotti*, Julian Thijssen*, Alexander Mordvintsev, Thomas Höllt,
Baldur van Lew, Boudewijn P.F. Lelieveldt, Elmar Eisemann and Anna Vilanova

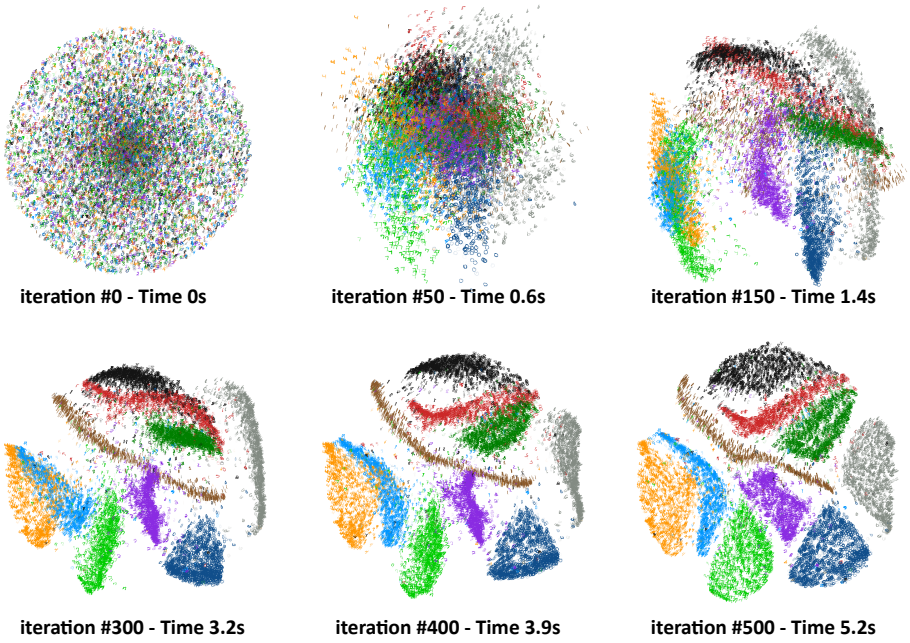

Fig. 1: Evolution of the t-SNE embedding for the MNIST dataset. The optimization is performed in only a few seconds while running in a web browser and providing progressive updates. Previous implementations require tens of minutes to run in multithreaded C++ programs. CUDA implementations exists, but require NVIDIA GPUs and do not run in the browser. The example can be run at the following link `https://nicola17.github.io/tfjs-tsne-demo/`

Abstract—In recent years the t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm has become one of the most used and insightful techniques for exploratory data analysis of high-dimensional data. It reveals clusters of high-dimensional data points at different scales while only requiring minimal tuning of its parameters. However, the computational complexity of the algorithm limits its application to relatively small datasets. To address this problem, several evolutions of t-SNE have been developed in recent years, mainly focusing on the scalability of the similarity computations between data points. However, these contributions are insufficient to achieve interactive rates when visualizing the evolution of the t-SNE embedding for large datasets. In this work, we present a novel approach to the minimization of the t-SNE objective function that heavily relies on graphics hardware and has linear computational complexity. Our technique decreases the computational cost of running t-SNE on datasets by orders of magnitude and retains or improves on the accuracy of past approximated techniques. We propose to approximate the repulsive forces between data points by splatting kernel textures for each data point. This approximation allows us to reformulate the t-SNE minimization problem as a series of tensor operations that can be efficiently executed on the graphics card. An efficient implementation of our technique is integrated and available for use in the widely used Google TensorFlow.js, and an open-source C++ library.

Index Terms—High Dimensional Data, Dimensionality Reduction, Progressive Visual Analytics, Approximate Computation, GPGPU

1 Introduction

Understanding how data points are arranged in a high-dimensional space plays a crucial role in exploratory data analysis [39]. In recent years, non-linear dimensionality reduction techniques became powerful tools for mining knowledge from data, such as for the discovery of clusters. In the field of data visualization, these techniques are used for reducing the dimensionality to two or three dimensions in order to make visualization possible. Specifically, the algorithms preserve certain characteristics of the data, such as the local neighborhoods. This is effective due to the fact that most of the real-world data satisfy the "manifold hypothesis", i.e., they lie on low-dimensional manifolds embedded in high-dimensional space.

The t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm [42] has become one of the state-of-the-art non-linear dimensionality reduction methods for visual analysis of high-dimensional

- *Nicola Pezzotti and Alexander Mordvintsev are with Google AI, Zürich, Switzerland.*
- *Nicola Pezzotti, Julian Thijssen, Thomas Höllt, Boudewijn P.F. Lelieveldt, Elmar Eisemann and Anna Vilanova are with the Delft University of Technology, Delft, The Netherlands.*
- *Thomas Höllt, Baldur van Lew and Boudewijn P.F. Lelieveldt are with the Leiden University Medical Center, Leiden, The Netherlands.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2019.2934307, IEEE Transactions on Visualization and Computer Graphics

data. It has been successfully applied to different domains, such as life sciences [2, 4, 25], the comprehension of machine-learning models and to human-driven supervision [17, 28, 33]. The t-SNE algorithm can be separated in two computation modules; first it computes the similarities of the high-dimensional points as a joint probability distribution and, second, it minimizes the Kullback–Leibler (KL) divergence [21], which measures the similarity between the data distribution in the high-dimensional space and the low-dimensional space.

The gradient of the KL divergence can be interpreted as a summation of attractive and repulsive forces between points, which makes the minimization process very similar to an N-body simulation [1]. The memory and computational complexity of the algorithm is $O\left(N^2\right)$, where N is the number of data points. Interactive computation times are essential in an interactive visual exploration solution, and in consequence much research effort has been spent on improving its computational and memory complexity.

While many works focused on improvement of the similarity computation [27,32,34,38,41], only limited effort has been spent on improving the minimization algorithm employed for the creation of the embedding [19, 27, 41]. Barnes-Hut-SNE (BH-SNE) was proposed by van der Maaten [41]. It makes use of the Barnes-Hut algorithm for N-body simulations [3] to approximate the repulsive forces between the data points. Repulsive forces change during minimization, since they depend on the data points position in the low-dimensional embedding space. Despite the improvements the computational costs remain high for large amounts of data points.

In this work, we focus on the minimization of the objective function, i.e., the KL-divergence, for the creation of the embedding. We observe that the heavy tail of the Student's t-distribution used by t-SNE makes the application of an N-body simulation not particularly effective. We propose a paradigm shift from point-to-point computation to a field-based computation of the embedding by reformulating the gradient of the objective function as a function of scalar and vector fields combined with tensor operations.

Our technique has linear computational and memory complexity, $O(N)$, and is suitable for implementation in a GPGPU fashion, providing considerably better computation times compared to the current state of the art. It also allows us to implement a version for the browser and desktop that minimizes the objective function for standard datasets in a matter of seconds, potentially enabling the development of more advanced web-based analytics solutions.

The contribution of our work is twofold:

- A linear complexity minimization of the t-SNE objective function. Specifically, we
  - approximate the repulsive forces between data points with a GPGPU approach relying on texture splatting
  - adopt a tensor-based computation of the objective function's gradient.
- An efficient implementation of our approach is released as part of Google's TensorFlow.js library and as part of the C++ HDI library. Our implementation is not only several orders of magnitude faster than the Barnes-Hut-SNE, but we demonstrate that it minimizes the objective function more effectively in addition to having better high-dimensional neighbor preservation.

The rest of the paper is structured as follows. In the next section, we provide a theoretical primer on the t-SNE algorithm that is needed to understand the related work (Section 3) and our contributions (Section 4). In Section 5, we provide details regarding our implementations. Finally, in Section 6, we compare our technique to BH-SNE, t-SNE-CUDA and the original t-SNE. We show the performance and accuracy improvements over these techniques using publicly available high-dimensional datasets.

2 t-SNE

In this section, we provide an introduction to the t-SNE [42] algorithm, which is essential to understand the related work and our contribution. The t-SNE algorithm interprets the overall distances between data points in the high-dimensional space as a symmetric joint probability distribution P that encodes their similarities. Likewise a joint probability distribution Q is computed that describes the similarity in the low-dimensional space. The goal is to achieve a representation, referred to as an *embedding*, in which Q faithfully represents P. This means that the embedding preserves the local neighborhoods of the high-dimensional data points. At the same time, the low-dimensional embedding only has two or three dimensions, which can easily be visualized.

This objective is achieved by optimizing the positions of the points in the low-dimensional embedding to minimize the cost function C given by the Kullback–Leibler, KL, divergence between the joint-probability distributions P and Q. Intuitively, points in the embeddings are moved in an iterative fashion, such that the embedding similarities encoded by Q become more closely matched to the similarities in the high-dimensional space encoded by P.

In more detail, given two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ in a high-dimensional dataset $X = \{\mathbf{x}_1 ... \mathbf{x}_N\}$, the probability p_{ij} models the similarity of these points in high-dimensional space. q_{ij} models the similarity in the low-dimensional embedding of the corresponding points $\mathbf{y}_i$ and $\mathbf{y}_j$. The cost function C is formulated as follows:

$$C(P,Q) = KL(P||Q) = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} p_{ij} \ln\left(\frac{p_{ij}}{q_{ij}}\right), \tag{1}$$

where KL measure the mismatch between Q and P. Similarities between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ in the high-dimensional space are represented by p_{ij}. More specifically, for each point $\mathbf{x}_i$, a Gaussian kernel is centered on the point and used to compute the probability that the other point is a neighbor. The variance σ_i of the kernel is defined according to the local density in the high-dimensional space, and p_{ij} is computed as follows:

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

$$\text{where} \quad p_{j|i} = \frac{\exp(-(||\mathbf{x}_i - \mathbf{x}_j||^2)/(2\sigma_i^2))}{\sum_{k\neq i}^{N} \exp(-(||\mathbf{x}_i - \mathbf{x}_k||^2)/(2\sigma_i^2))} \tag{3}$$

$p_{j|i}$ can be seen as a relative measure of similarity for the point $\mathbf{x}_i$ and all the points $\mathbf{x}_j$ in its local neighborhood. The effective number of neighbors considered for each data point is derived by the perplexity value μ, which is a user-defined parameter. Consequently, the value of σ_i is chosen such that for a fixed perplexity μ and for each i it satisfies:

$$\mu = 2^{-\sum_j^N p_{j|i} \log_2 p_{j|i}} \tag{4}$$

A *Student's t-Distribution* with one degree of freedom is used to compute the joint probability distribution in the low-dimensional embedding Q, where the positions of the data points should be optimized. Q plays a similar role for the points in the low-dimensional space, as P does for the high-dimensional space. It encodes the similarities given the neighborhood information. In the embedding space the dispersion of the distribution (i.e., the Student's t-Distribution) is constant. Given two low-dimensional points $\mathbf{y}_i$ and $\mathbf{y}_j$, the probability q_{ij} is given by:

$$q_{ij} = \left((1 + ||\mathbf{y}_i - \mathbf{y}_j||^2)Z\right)^{-1} \tag{5}$$

$$\text{with} \quad Z = \sum_{k=1}^{N} \sum_{l\neq k}^{N} (1 + ||\mathbf{y}_k - \mathbf{y}_l||^2)^{-1} \tag{6}$$

The goal of a t-SNE optimization is to move randomly initialized points $\mathbf{y_i}$ in the embedding, such that the distribution Q is as close as possible to the distribution P. Intuitively, when Q matches P, the neighborhoods in the low-dimensional space match the high-dimensional

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2019.2934307, IEEE Transactions on Visualization and Computer Graphics

counterparts. This result is obtained by minimizing a cost function C which is defined as the Kullback–Leibler divergence between P and Q. The gradient of C has an analytical solution and indicates the change in position of the points $\mathbf{y_i}$. It is given by:

$$\frac{\delta C}{\delta \mathbf{y}_i} = 4(F_i^{\text{attr}} - F_i^{\text{rep}}) \tag{7}$$

$$= 4(Z\sum_{j\neq i}^{N} p_{ij}q_{ij}(\mathbf{y}_i - \mathbf{y}_j) - \sum_{j\neq i}^{N} q_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)) \tag{8}$$

The optimization is based on gradient descent. For each iteration, the gradient is used to update the position of the data points in the embedding. The gradient descent can be seen as an *N-body simulation* [1], where each data point exerts an attractive and a repulsive force (F_i^{attr} and F_i^{rep}) on all other points.

3 Related Work

We now present the work that has been done to improve the computation of t-SNE embeddings in terms of quality and scalability. Van der Maaten proposed the Barnes-Hut-SNE (BH-SNE) [41], which reduces the complexity of the algorithm to $O(N\log(N))$ for both the similarity computations and the objective function minimization. More specifically, in the BH-SNE approach the similarity computations are seen as a k-nearest neighborhood graph computation problem, which is obtained using a Vantage-Point Tree [45]. The minimization of the objective function is then seen as an N-body simulation, which is solved by applying the Barnes-Hut algorithm [3].

In our previous work [34], we observed that the computation of the k-nearest neighborhood graph for high-dimensional spaces using the Vantage-Point Tree is affected by the curse of dimensionality, limiting the efficiency of the computation. To overcome this limitation, we proposed the Approximated-tSNE (A-tSNE) algorithm [34], where approximated k-nearest neighborhood graphs are computed using a forest of randomized KD-trees [29]. Moreover, A-tSNE adopts the novel Progressive Visual Analytics paradigm [11,36], allowing the user to observe the evolution of the embedding during the minimization of the objective function. This solution enables a user-driven early termination of the algorithm. t-SNE-CUDA [7] is a CUDA implementation of the Approximated-tSNE algorithm. For computing the high-dimensional neighborhood, it uses the GPU library FAISS [16]. A tree structure based on the BH-SNE is implemented in CUDA to compute the repulsive forces. While the technique allows for a fast computation of the embedding, the application is limited to NVIDIA hardware, greatly limiting its application. Furthermore, like BH-SNE, the resulting embedding remains an approximation of the t-SNE embedding.

A similar observation on the benefit of using approximated computations was later made by Tang et al. that led to the development of the LargeVis technique [38]. LargeVis uses random projection trees [9] followed by a kNN-descent procedure [10] for the computation of the similarities and a different objective function that is minimized using a Stochastic Gradient Descent approach [18]. Despite the improvements, both the A-tSNE and LargeVis tools still suffer from long computation times during the optimization that hinder interaction for large data sets. Better performance is achieved by the UMAP algorithm [27], which provides a different formulation of the dimensionality-reduction problem as a cross-entropy minimization between topological representations. Computationally, UMAP follows LargeVis very closely and adopts a kNN-descent procedure [10] and stochastic gradient-descent minimization of the objective function.

A different approach is taken in the Hierarchical Stochastic Neighbor Embedding algorithm (HSNE) [32]. HSNE efficiently builds a hierarchical representation of the manifolds and embeds only a subset of the initial data that represents an overview of the available manifolds. The user can "drill-in" the hierarchy by requesting more detailed embeddings that reveal smaller clusters of data points. While HSNE allows scalability of the analysis to large data sets by the generation and user-guided exploration of multiple embeddings, it does not address the acceleration of the computation of single embeddings.

The techniques presented so far do not take advantage of the dimensionality of the target domain. As a matter of fact, t-SNE is mostly used for data visualization in two-dimensional scatterplots, while the techniques introduced in this section so far are general and can be used in target domains of any dimensionality. Based on this observation, Kim et al. introduced the PixelSNE technique [19], where the points are not embedded in a continuous 2D space, but rather in a discretized space corresponding to the pixels used to display the scatterplot. The optimization is performed using an N-body simulation approach, which is similar to the one employed by BH-SNE. In order to compute embeddings that faithfully preserve high-dimensional neighborhoods, a large number of pixels must be used, often much larger than the display's resolution. In addition, it hampers the scalability of the technique, requiring many hours to compute embeddings containing more than a million points.

In our work, we take advantage of the two-dimensional domain in which the embedding resides and we propose an efficient way to minimize the t-SNE objective function. Contrary to PixelSNE, we only discretize the two-dimensional space for the computation of the repulsive forces presented in Equation 8. We developed a linear-complexity approach implemented using GPGPU as a desktop and client-side browser application. This is an improvement over t-SNE-CUDA, which can only be run on NVIDIA GPUs. Even though their computation of the embedding is faster, our technique produces embeddings that match more closely to the high-dimensional space. Compared to BH-SNE and PixelSNE, our technique computes embeddings with more than a million points in just a few minutes instead of several hours, while providing better preservation of high-dimensional similarities.

4 Linear complexity t-SNE minimization

In this section, we present our approach to minimizing the t-SNE objective function as presented in Equation 1. The main idea consists in rewriting the gradient presented in Equation 7 such that it relies on a scalar field $\mathscr{S}$ and a vector field $\mathscr{V}$ in the 2D embedding domain. These fields can be computed in linear time on the GPU and queried in constant time. Therefore, the complexity of the algorithm is reduced from quadratic to linear.

4.1 Field-based computation of the gradient

The gradient of the objective function has the same form as in regular t-SNE:

$$\frac{\delta C}{\delta \mathbf{y}_i} = 4(\hat{F}_i^{\text{attr}} - \hat{F}_i^{\text{rep}}), \tag{9}$$

with attractive and repulsive forces acting on every point $\mathbf{x}_i \in X$. We denote the forces with a $\wedge$ to distinguish them from their original counterparts. We rewrite the equation of the gradient in the form of a scalar field $\mathscr{S}$ and a vector field $\mathscr{V}$:

$$\mathscr{S}(\mathbf{p}) = \sum_{i}^{N} \left(1 + ||\mathbf{y}_i - \mathbf{p}||^2\right)^{-1}, \mathscr{S} : R^2 \Rightarrow R \tag{10}$$

$$\mathscr{V}(\mathbf{p}) = \sum_{i}^{N} \left(1 + ||\mathbf{y}_i - \mathbf{p}||^2\right)^{-2} (\mathbf{y}_i - \mathbf{p}), \mathscr{V} : R^2 \Rightarrow R^2 \tag{11}$$

Intuitively, $\mathscr{S}$ represents the density of the points in the embedding space, according to the Student's t-distribution, and it is used to compute the normalization of the joint probability distribution Q. An example of the field $\mathscr{S}$ is shown in Figure 2b. The vector field $\mathscr{V}$ represents the directional repulsive force applied to the entire embedding space. An example of $\mathscr{V}$ is presented in Figures 2c and d, where the horizontal and vertical gradient components are visualized separately. If a point in the embedding resides in the red area of Figure 2c, it will be pushed a certain amount to the right in the current iteration of the gradient descent, while a point in the blue area will be pushed to the left. Similarly for the vertical component, see Figure 2d, a point will be pushed either up, for red areas, or down for blue ones. We describe

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2019.2934307, IEEE Transactions on Visualization and Computer Graphics

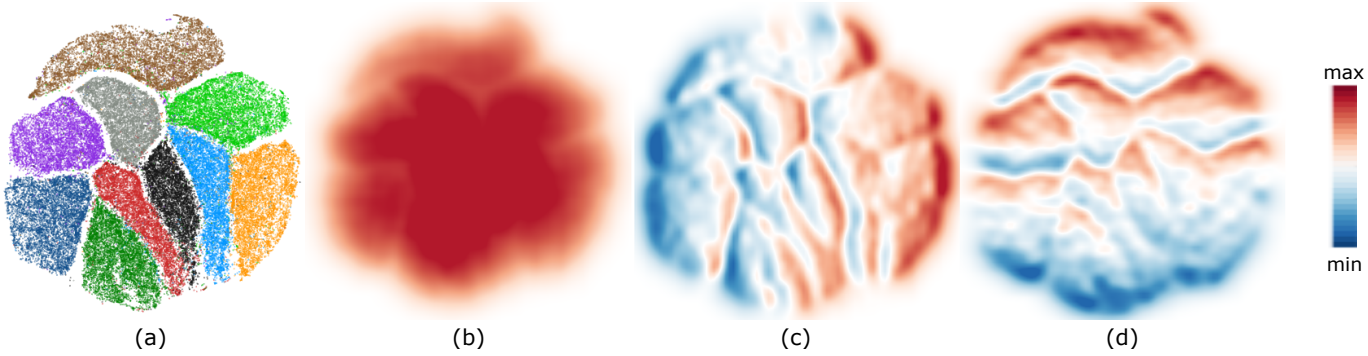

Fig. 2: **Fields** used in our approach. (a) The MNIST dataset contains images of handwritten digits and is embedded in a 2-dimensional space. The minimization of the objective function is computed in linear time by making use of a scalar field $\mathcal{S}$ (b) and a 2-dimensional vector field $\mathcal{V}$, where (c-d) show the horizontal and vertical components respectively. The fields are computed on the GPU by drawing properly designed mathematical kernels using the additive blending function of the rendering pipeline. The rest of the minimization is treated as a series of tensor computations that are computed on the GPU.

the construction of $\mathcal{S}$ and $\mathcal{V}$ in Section 4.2. For now, we assume these fields are given, and we present how the gradient of the objective function is derived from $\mathcal{S}$ and $\mathcal{V}$.

For the attractive forces, we adopt the restricted neighborhood contribution as presented in the Barnes-Hut-SNE technique [41]. The rationale of this approach is that, by imposing a fixed perplexity on the Gaussian kernel, only a limited number of neighbors effectively apply an attractive force on any given point (see Equations 3 and 4). Therefore we limit the number of contributing points to some multiple of the chosen perplexity. This approach reduces the computational and memory complexity of the computation of the attractive forces to $O(N)$, since the size of the neighborhood k is several orders of magnitude lower than N, $k \ll N$.

$$\hat{F}_i^{\text{attr}} = \hat{Z} \sum_{l \in \text{kNN}(i)} p_{il} q_{il} (\mathbf{y}_i - \mathbf{y}_l) \tag{12}$$

The computation of the normalization factor Z, as it is presented in Equation 6, has computational complexity $O\left(N^2\right)$. In our approach, we compute $\hat{Z}$ by consulting the scalar field $\mathcal{S}$ in constant time, giving us a complexity of $O(N)$.

$$\hat{Z} = \sum_{l=1}^{N} (\mathcal{S}(\mathbf{y}_l) - 1) \tag{13}$$

Note that the formulation of Z and $\hat{Z}$ is identical but, since $\mathcal{S}$ is computed in linear time, computing $\hat{Z}$ also has linear complexity. $\hat{Z}$ does not depend on the point $\mathbf{y}_i$ for which we are computing the gradient. Therefore, $\hat{Z}$ needs to be computed just once, cached, and then used at each iteration of the gradient descent for all points.

The repulsive force assumes the following form

$$\hat{F}_i^{\text{rep}} = \mathcal{V}(\mathbf{y}_i)/\hat{Z}, \tag{14}$$

where the value of the vector field $\mathcal{V}$ in the location identified by the coordinates $\mathbf{y}_i$ is normalized by $\hat{Z}$. Similar to $\hat{Z}$, $\hat{F}^{\text{rep}}$ has an equivalent formulation as F^{rep} but with computational and memory complexity equal to $O(N)$. So far, we assumed that $\mathcal{S}$ and $\mathcal{V}$ are computed in linear time and queried in constant time. In the next section, we present how the rasterization pipeline is used to compute an approximation of the $\mathcal{S}$ and $\mathcal{V}$ fields. In Section 5, two ways to implement the proposed approach are given.

4.2 Computation of supporting fields

Our approach to the computation of the fields resembles an approach used for Kernel Density Estimation [35], which has applications in visualization [22] and non-parametric clustering [13]. In this setting, given a number of points, the goal is to estimate a two-dimensional probability density function. This is achieved by superimposing a

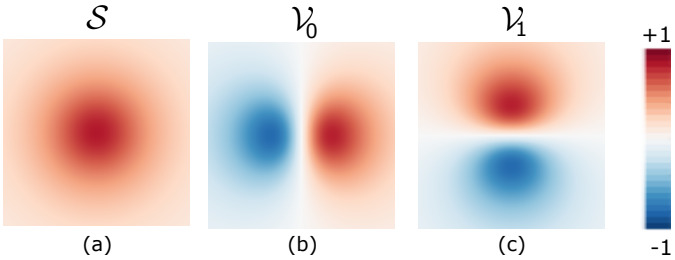

Fig. 3: **Functions** drawn over each embedding point to approximate the scalar field $\mathcal{S}$ and the 2-dimensional vector field $\mathcal{V}$.

Gaussian kernel, whose σ has to be estimated, over every data point. Summing the contributions of all points in a given location or pixel in the embedding gives us the probability density function in a given location.

In KDE methods, the 2D kernel density is estimated efficiently on the GPU because of the quasi-limited support of the kernels, i.e., having values almost equal to zero if they are sufficiently far away from the origin. A good approximation of the density function is then achieved by drawing a quad at the location of each sample, which contains a precomputed texture or evaluates the kernel for each covered pixel [5, 22]. By using additive blending, i.e., by summing the values in every pixel, the resulting output approximates the desired density function.

In our context, we want to compute $\mathcal{S}$ and $\mathcal{V}$ as shown in equations 10 and 11. These equations can also be seen as a summation of kernels S and V as defined in the following equations:

$$\mathcal{S}(\mathbf{p}) = \sum_i^N S(\mathbf{y}_i - \mathbf{p}), \quad S(\mathbf{d}) = \left(1 + ||\mathbf{d}||^2\right)^{-1} \tag{15}$$

$$\mathcal{V}(\mathbf{p}) = \sum_i^N V(\mathbf{y}_i - \mathbf{p}), \quad V(\mathbf{d}) = \left(1 + ||\mathbf{d}||^2\right)^{-2} (\mathbf{d}) \tag{16}$$

The presented kernels S and V are stored in a texture and are presented in Figure 3. The kernels have a limited function support, making it indeed very similar to the Kernel Density Estimation case discussed before. As the fields $\mathcal{S}$ and $\mathcal{V}$ are a summation of the aforementioned kernels, we can compute an approximation of the fields by additively rendering these per-point kernel textures at the locations of each of the points in the embedding.

The resulting 3-channel texture, an example of which is presented in Figures 2b-d, represents the scalar field $\mathcal{S}$ and the vector field $\mathcal{V}$. Fetching the value of $\mathcal{S}$ and $\mathcal{V}$ for a point $\mathbf{y}_i$ then corresponds to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2019.2934307, IEEE Transactions on Visualization and Computer Graphics

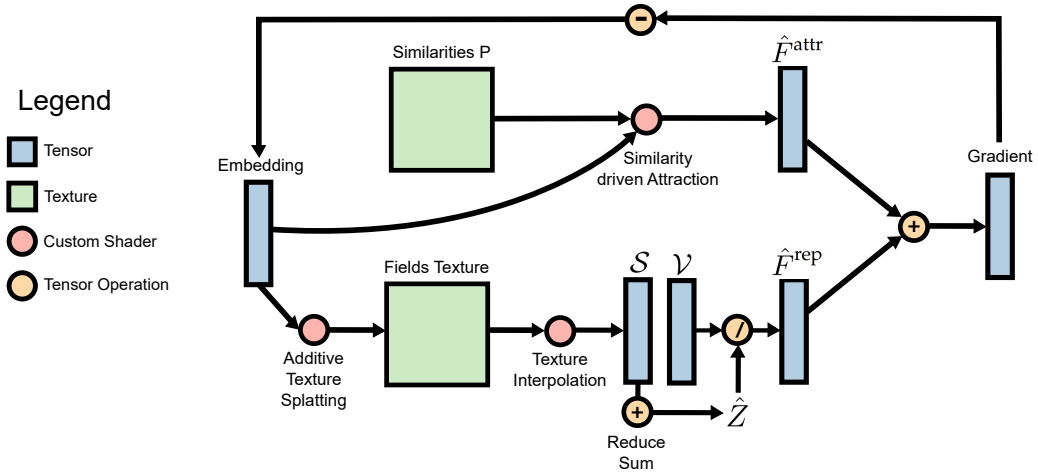

Fig. 4: **Computational workflow** of our approach. On the lower side of the chart, the computation of the repulsive forces is presented. The fields texture is generated by the additive texture splatting presented in Section 5.1.2. The values of $\mathscr{S}$ and $\mathscr{V}$ are obtained through texture interpolation and are used to compute the repulsive forces. The attractive forces are computed in a custom shader that takes as input the similarities P and the embedding. The gradient of the objective function is then computed using both forces and is used to update the embedding.

extracting the interpolated value at the point's position in the field textures.

Contrary to the Kernel Density Estimation case, where the size of the quads changes according to the σ chosen for the Gaussian kernel, our functions must have a fixed support in the embedding space. This is dictated by the fact that we are optimizing Equation 1, a change of the quad size corresponds to a change in the low-dimensional distribution characterizing the points. Therefore, the resolution of the texture influences the quality of the approximation but not the overall shape of the fields. To achieve linear complexity, we define the resolution of the aggregate field texture according to the size of the embedding. The number of pixels that are covered by the textures presented in Figure 3 is kept constant. This is achieved by changing the size of the target texture in the embedding space. A ratio ρ between the diameter of the embedding and the texture resolution is fixed. Hence, every data point updates the value of a constant number of pixels in the target texture equal to ρ^2. This solution leads to $O(N\rho^2)$ complexity for the computation of the fields, and we empirically found $\rho = 0.5$ to be a good compromise between the fidelity of the resulting fields and the computation time required. Since $\rho^2 \ll N$, the resulting computational complexity is $O(N)$. Note that, by being adaptive to the texture size, no parameter tuning is required. A potential limitation is the maximum embedding size as defined by the OpenGL standard. In practice, this does not pose a limit since the embeddings size does.

5 Implementations

In this section we explain how the ideas presented in the previous section are implemented both for the browser as part of TensorFlow.js and for the desktop as part of the open source High-Dimensional Inspector (HDI) library [31]. Two different approaches are presented: one that makes use of the rasterization pipeline, and one that uses compute shaders.

5.1 Rasterization Approach

In this section, we present an implementation that heavily makes use of the rasterization pipeline of modern GPUs. Rasterization is the task of converting a series of geometric primitives, most commonly triangles, into a series of pixels that form a raster image. Contrary to the common application of rasterization in computer graphics, i.e., rendering of geometric scenes, here we associate each pixel with an atomic computation used for minimizing the t-SNE loss function. These are the computation of the attractive forces given the similarity distribution P (Section 5.1.1), the computation of the fields used for computing the repulsive forces (Section 5.1.2) and subsequently the updating of the embedding (Section 5.1.3).

5.1.1 Attractive Forces

Computation of the attractive forces, shown in the upper portion of Figure 4, is performed by measuring the sum of the contribution of every neighboring point in the high-dimensional space. The neighborhoods are encoded in the joint probability distribution P which is stored in a sparse matrix. P can be computed ahead of time, for example using an approximated k-nearest-neighborhood algorithm [9, 10, 29] or by the HSNE technique [32]. We use existing techniques here, and do not provide any contribution.

5.1.2 Repulsive Forces

We achieve linear complexity for the computation of the repulsive forces by making use of the rasterization pipeline innate in graphics cards. For the browser implementation we make use of the WebGL API and for the desktop implementation we use standard OpenGL.

In order to form the field textures we start with a randomly initialized t-SNE embedding. Centered on each of the points in the embedding, a quad is rendered. We apply a texture to the quad whose R color channel contains $S(\mathbf{p})$ from Equation 15 and whose G and B color channels contain $V(\mathbf{p})$ in each dimension from Equation 16. By enabling additive blending these splatted textures will add up to an approximation of the $\mathscr{S}$ and $\mathscr{V}$ fields. The approximated fields are stored in another floating-point RGB texture whose resolution is proportional to the size of the embedding space. The ratio between the two is defined by the parameter ρ introduced in Section 4.2. The degree of approximation is controlled by the resolution of the aggregate field texture and the resolution of the kernel texture.

To query the field values for a specific point in the embedding, we sample the field value at the point's position using bilinear texture interpolation. This operation is natively supported in the GPU and very efficient. The normalization factor $\hat{Z}$ is obtained by summing all the elements in the tensor with the interpolated values of $\mathscr{S}$. This summation is performed as a reduction operation on the graphics card. Note that $\hat{Z}$ is computed once and cached, hence Equation 14 is computed by simply dividing the interpolated field value by the cached $\hat{Z}$.

5.1.3 Updating the points

The remaining computational steps are computed as tensor, i.e., matrix, operations as defined in toolkits like TensorFlow.js. $\hat{F}^{\text{rep}}$ is obtained by dividing the interpolated values of $\mathscr{V}$ by $\hat{Z}$, and the gradient of the objective function is obtained by adding the attractive forces $\hat{F}^{\text{attr}}$. The gradient is then applied to the embedding modifying the position of the points according to the gradient. Figure 4 shows an overview of our approach. Green squares represent textures containing the computed fields or the similarity matrix P, while blue rectangles represent tensors.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2019.2934307, IEEE Transactions on Visualization and Computer Graphics

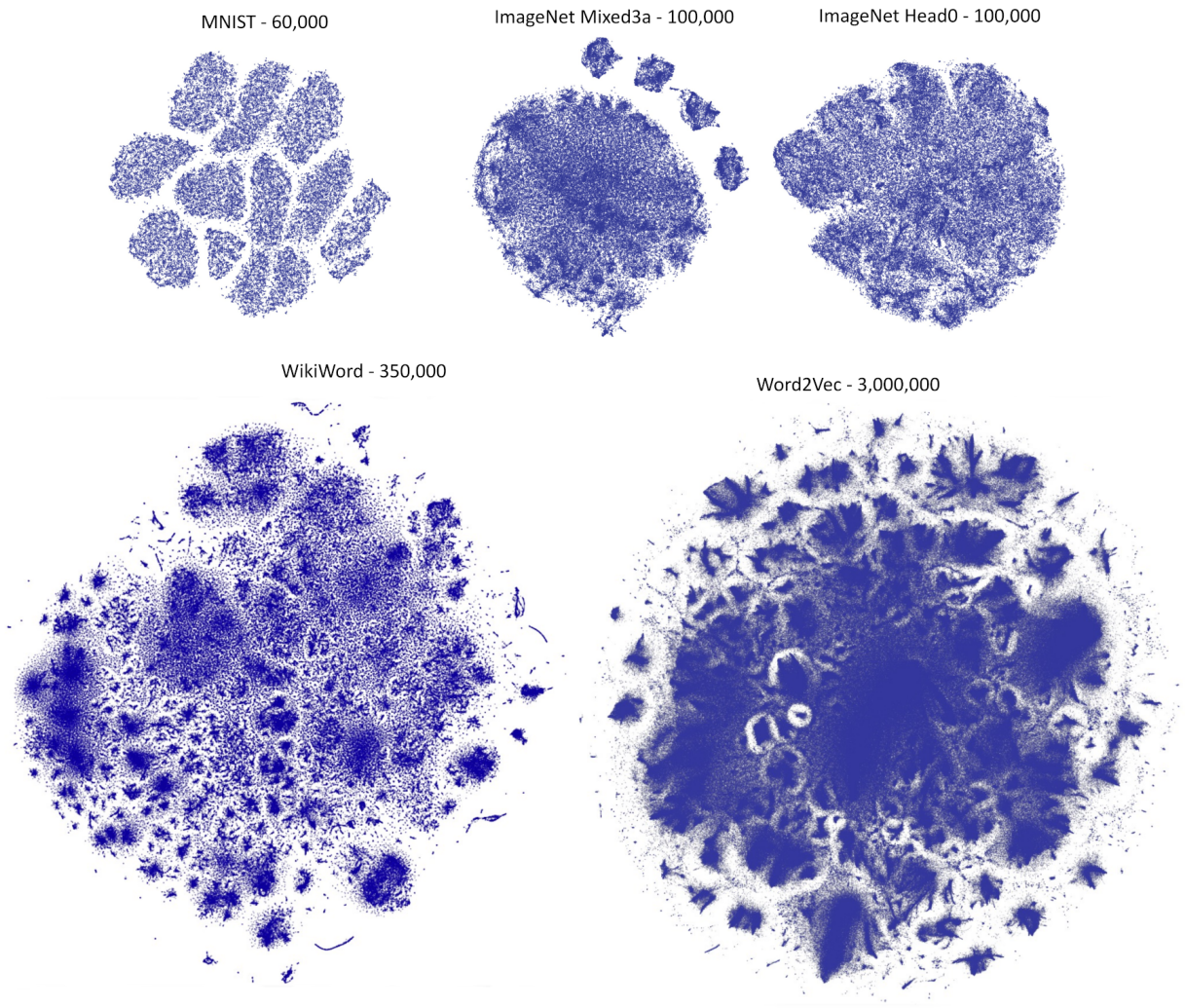

Fig. 5: **Embeddings** of the MNIST, ImageNet Mixed3a, ImageNet Head0, WikiWord and Word2Vec datasets generated by our technique.

Operations are represented by circles. More specifically, red circles are custom operations that are implemented specifically for our technique. Orange circles are tensor operations that are commonly available in TensorFlow.js or in the HDI library.

5.2 Compute Shader Approach

Implementations of our approach are available for both the web and desktop. These implementations are broadly applicable due to their limited feature requirements. However, as the computation of the algorithm is essentially reduced to a series of tensor operations, it lends itself very well to execution using one of the GPGPU APIs available. In the rasterization approach, many splats might overlap with each other. In particular, when the function support of the t-distribution is increased for more accurate embeddings, this simultaneously results in more overlapping splats. With additive blending enabled, this results in a high degree of overdraw, which can be quite costly. For this reason we have developed another implementation of the previously described algorithm. Instead of splatting textures to obtain the fields, here, we calculate the fields in a compute shader in the following manner.

For each pixel in the output field we calculate the influence of per-point kernels on this pixel. If the point lies further away from the current pixel in embedding space than the given function support, the point is ignored. The complexity of this operation is $O(NPx)$ where Px represents the number of pixels used for the output field. In practice, our solution behaves very linearly, since the maximum number of pixels affected is much lower than the number of points in reasonably sized data sets. This means, that the function support can be unbounded with negligible loss of performance, thereby resulting in even more accurate embeddings. This can also be done in the rasterization approach, however, it would result in extreme overdraw and have a significant impact on performance.

6 Evaluation

In order to assess the efficacy of the proposed technique we evaluate the computational costs and quality of the embedding using three metrics. First, we record the execution time of the minimization process over 1000 iterations. Secondly, we evaluate the quality of the resulting embedding by using the reached Kullback–Leibler divergence. Kullback–Leibler is the objective function of the t-SNE algorithm. This metric shows how well the objective function is optimized by the different techniques. We also compute the Nearest-Neighbor Preservation (NNP) metric as described by Venna et al. [44] and implemented by Ingram and Munzner [15]. It measures how well small neighborhoods in the high-dimensional space are preserved during the dimensionality reduction. The main benefit of such a metric is its independence from the objective function optimized by the t-SNE algorithm. In order to measure the NNP accurately it is important that the gradient descent has fully converged. We chose 1000 iterations for the MNIST and ImageNet datasets and 5000 iterations for the WikiWord and Word2Vec datasets to guarantee full convergence for the different data sizes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2019.2934307, IEEE Transactions on Visualization and Computer Graphics

We compare the results of our technique (i.e., GPGPU-SNE) with the results obtained from the Barnes-Hut-SNE [3] and the t-SNE algorithm without computational improvements [42]. Both implementations are written in C++, support multi-threaded computations and are openly available in the High-Dimensional-Inspector (HDI) library [31]. For Barnes-Hut-SNE, we provide results for two different values of its θ parameter. This parameter controls the trade-off between speed and accuracy of the algorithm. A value of $\theta = 0.5$ sacrifices accuracy slightly for the benefit of a significant performance boost, and is often chosen as the default value. A value of $\theta = 0.1$ prioritizes generating embeddings closer to those produced by original t-SNE, but at considerable execution time cost. Moreover, we provide a comparison with the t-SNE-CUDA algorithm [7] for a value of $\theta = 0.0$ and 0.5.

We expect that our implementation outperforms BH-SNE in time as well as quality of the embeddings. Our approach is fundamentally a different method of acceleration compared to t-SNE-CUDA. Our method does not rely on the CUDA API and can therefore be used to create embedding in a web-browser. Concerning performance, we expect t-SNE-CUDA to be similar or better concerning the computational costs, but lower in quality since it is an acceleration based on the approximation of BH-SNE.

6.1 Datasets

We have chosen five commonly used datasets to illustrate the applicability of our technique to both small and large amounts of high-dimensional data. First, we use the **MNIST** dataset. It consists of 60k labeled grayscale images of handwritten digits (compare Figure 2a). Each image is represented as a 784 dimensional vector, corresponding to the gray values of the pixels in the image. The MNIST data is often used to validate non-linear dimensionality reduction techniques. As a matter of fact, it clearly contains 10 different manifolds, one for each digit. Moreover, the manifolds are non-linear, hence linear dimensionality-reduction techniques such as PCA are not able to reconstruct the manifolds.

Table 1: **Datasets** used for the evaluation.

Dataset	Number of points	Number of dimensions
MNIST-60000	60000	768
WikiWord	350000	300
GoogleNews	3000000	300
ImageNet Mixed3a	100000	256
ImageNet Head0	100000	128

The **WikiWord** and **GoogleNews** datasets contain words, which are associated with a vector representation. These vector representations are algorithmically generated by processing large text corpora, often through a deep neural network [24] and by requiring that words that occur in similar contexts share a similar representation. The shapes associated with each word present interesting characteristics for latent semantic analysis [23]. As an example, it is shown that simple summation and subtraction of the vectors representing the words $King - Man + Woman$, as produced by the GloVe model [30], is very similar to the vector representation associated with the word *Queen*. Non-linear dimensionality reduction is often used in systems for the analysis of such word representations [8, 12, 26].

Finally, we present two different datasets obtained by collecting the activations of different layers in a deep neural network (DNN) [24] on the validation set of the ImageNet dataset [20][1]. The resulting embeddings shed a light on the internal computations performed by the deep neural network, the Google Inception [37] in this case. Images, or image patches, that are close in the embedding are considered similar by the DNN [33]. Recently, an increasing number of web-based tools, like the Activation Atlas [6] or Tensorboard, have been proposed to better understand and improve DNNs through dimensionality reduction techniques such as t-SNE or UMAP.

[1]The datasets can be created for an arbritary activation layer using the following Colab Notebook: `https://colab.research.google.com/github/tensorflow/lucid/blob/master/notebooks/activation-atlas/activation-atlas-simple.ipynb`

6.2 Results

In Figure 6, we show the results of the experiments for the chosen datasets. All experiments are conducted on an Intel Core i7-4820K Processor, with 4 physical cores (8 threads) @ 3.70 Ghz. The machine has 16GB of DDR3 RAM, and an NVIDIA GeForce GTX Titan GPU with 2688 CUDA cores @ 837 Mhz and 6GB of GDDR5 memory. All experiments run fit in the main memory available and have no interaction with disk during the optimization process.

To better highlight the behaviour of the algorithms with increasing dataset sizes, we run the algorithm on a random subset of the data with a growing number of data points for each of the experiments. The first row of charts in Figure 6 shows the execution time of the various algorithms plotted against the number of data points in the subsampled dataset. Note that a logarithmic scale is used for both the vertical and horizontal axes.

Our technique significantly cuts back on execution time compared to Barnes-Hut-SNE and t-SNE. For the MNIST dataset, t-SNE takes two days to complete the iterations. BH-SNE with $\theta = 0.1$ takes one hour and with $\theta = 0.5$ takes around 8 minutes. While our technique computes the embedding in just 16 seconds. This is a reduction on the cost of the gradient descent in the range of orders of magnitude. For the other datasets it becomes infeasible to run the first two algorithms as they would take many days to execute. It is possible to run BH-SNE $\theta = 0.5$ on the WikiWord dataset, but the computation takes more than an hour, while our technique computes the embedding in a mere 35 seconds. t-SNE-CUDA outperforms our technique by a factor in the range of $x2$ to $x5$. This can be explained by the highly-optimized code enabled by the CUDA implementation.

The second row examines the KL-divergence of the final embeddings from their original high-dimensional counterparts. And the last row shows the Nearest Neighborhood Preservation of all the embeddings, presented as a precision/recall plot.

In comparison to other optimization methods our technique produces a better, i.e., lower KL-divergence at data sets of non-trivial size. A likely explanation for this is that as the datasets get larger, the domain of the embedding expands but this expansion is not linear in the number of points. Therefore, the embedding will get progressively more dense, which is unfavourable for the Barnes-Hut approximation, which is also used by the t-SNE-CUDA. Approximations of the forces applied by distant points will become coarser as more of them are lumped together. Consequently this lowers the accuracy of the algorithm. This results in embeddings where the objective function cannot be effectively minimized, hence resulting in lower nearest-neighbor preservation. This observation is confirmed by the results presented in the third row. A similar observation can be made for the t-SNE-CUDA algorithm. Here, even higher KL-divergence can be observed for lower numbers of data points in the embedding. Speed is traded in favour of quality in producing the final embedding.

In the last row of Figure 6, we present the nearest-neighbor preservation for the different data sets. For each point, we examine a neighborhood of k points in the high and low-dimensional space. For every value from $k = 1$ to $k = 30$ we compute the true positive T, defined as the points that belong to both neighborhoods. From this, we compute precision as T/k, while recall is defined as $T/30$. The values of precision and recall for each value of k form a precision/recall curve for every point. The precision/recall curve for the entire embedding is obtained by averaging the curves of every point in the dataset. Since t-SNE and BH-SNE with $\theta = 0.1$ take days to compute on these datasets, it becomes infeasible to calculate the metric for all datasets. We provide it for the MNIST dataset to give an indication of the relationship between the techniques. In addition, for the 3-million data point Word2Vec dataset calculating the metric would take more than a week. Therefore, we compute it on a 350k subset of the dataset, which also allows the curve for Barnes-Hut-SNE to be presented. We see that our technique has a significant advantage over the Barnes-Hut-SNE and t-SNE-CUDA algorithm, as it presents a high Precision/Recall curve

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2019.2934307, IEEE Transactions on Visualization and Computer Graphics

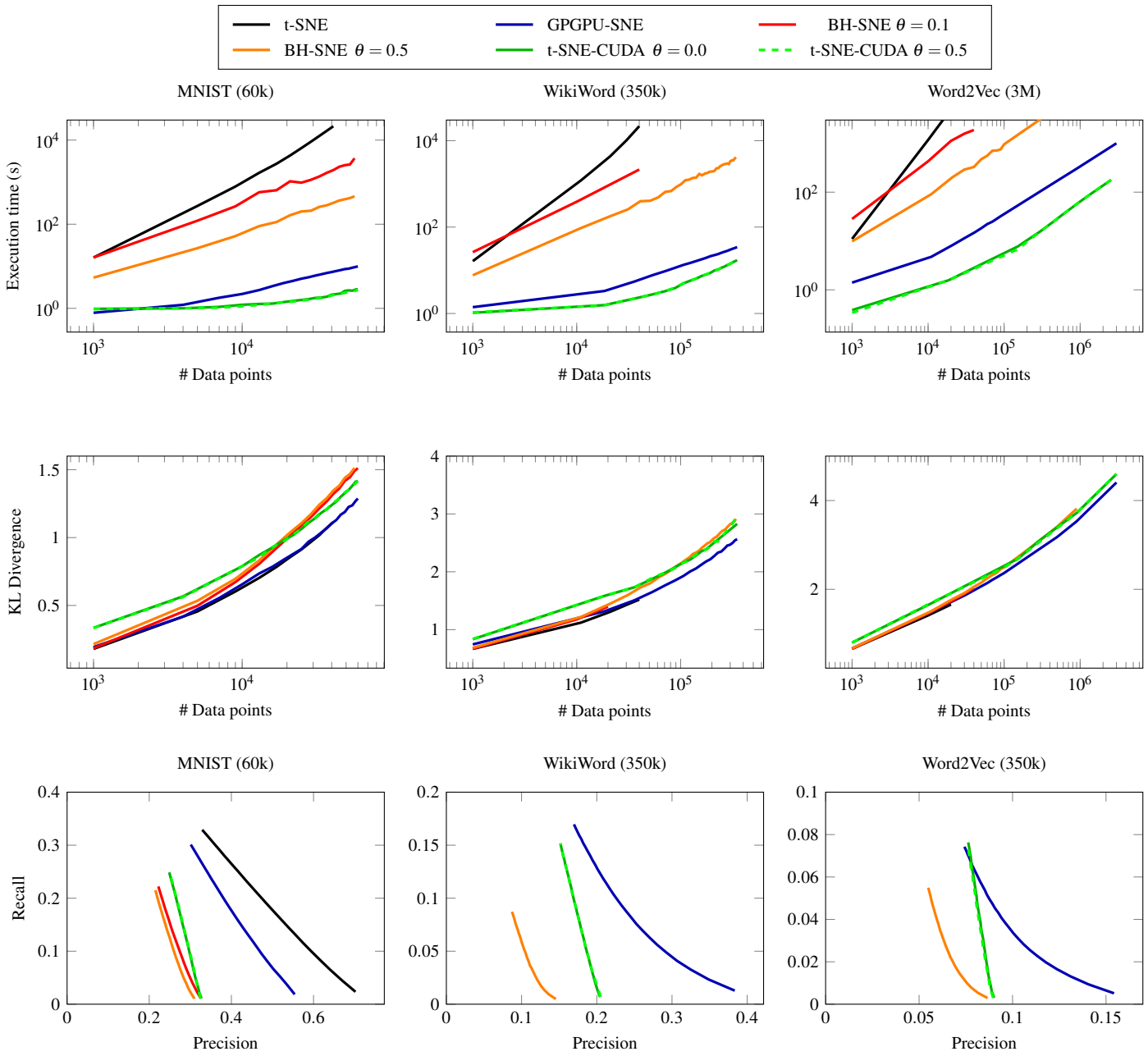

Fig. 6: **Results** of the experiments on the MNIST, WikiWord and Word2Vec datasets for the t-SNE, Barnes-Hut-SNE, t-SNE-CUDA and our approach. The first row shows the evolution of the execution time with increasingly bigger subsets of the dataset. The second row shows how well the objective function is fulfilled, while the third row shows the Nearest-Neighborhood Preservation (NPP). Our technique is up to two orders of magnitude faster than Barnes-Hut-SNE and provides higher quality embeddings compared to Barnes-Hut-based techniques.

in all measured datasets. Figure 7 shows the results on the ImageNet datasets for our technique, BH-SNE with $theta = 0.5$ and t-SNE-CUDA with $theta = 0.0$ and 0.5. The results confirm the previous analysis, showing that our technique beats the BH-SNE by almost two orders of magnitude. t-SNE-CUDA is faster by a factor of approximately $x3$ on the full dataset, requiring less than 4 seconds while our approach computes the embeddings in 11 seconds. Our solution, however, shows lower KL-divergence and better precision and recall than both BH-SNE and t-SNE-CUDA.

7 Conclusion

In this work, we presented a novel approach for the optimization of the objective function of t-SNE that scales to large datasets. We provided a reformulation of the gradient equations of the objective function that includes a scalar and a vector field. These fields represent the point density and the directional repulsive forces in the embedding space. Our approach relies on modern graphics hardware to efficiently compute these fields, obtaining linear complexity in the number of points compared to the quadratic complexity of the non-accelarated t-SNE.

In our experiments, we observe that our implementation outperforms the Barnes-Hut-SNE algorithm by several orders of magnitude. Besides the faster optimization, our technique is better at minimizing the objective function than all other acceleration methods, i.e., having a lower Kullback-Leibler divergence, and provides better Nearest-Neighbor Preservation. t-SNE-CUDA outperforms our method in computational times, but produces lower quality embeddings, and relies on NVIDIA GPUs, which limits its applicability.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2019.2934307, IEEE Transactions on Visualization and Computer Graphics

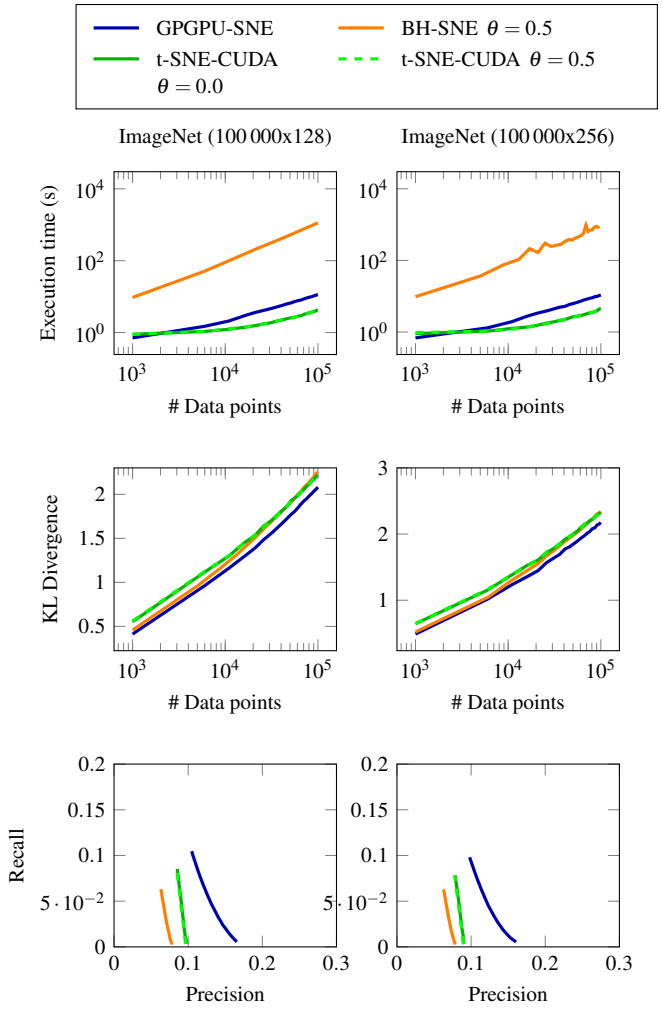

Fig. 7: **Results** of the experiments on the ImageNet datasets for Barnes-Hut-SNE, t-SNE-CUDA and our approach.

We provide two implementations of our technique. The first one is available in the High-Dimensional Inspector library. The library, which can be found at the following link `https://github.com/Nicola17/High-Dimensional-Inspector`, is a C++ library used by several visual-analytics applications such as Cytosplore [13, 14, 43]. The second implementation is released as part of TensorFlow.js and can be found on GitHub at the following address: `https://github.com/tensorflow/tfjs-tsne`.

As future work, we want to explore how our implementation can be integrated in Progressive Visual Analytics systems [11, 40], such as tools for the analysis of Deep Neural Networks. For example, the Embedding Projector [2], TensorBoard [3] and DeepEyes [33]. A limitation of the presented technique is that a graphics card is required in order to run the algorithm, which potentially restricts its applicability. In addition, our technique shares the intrinsic problems of t-SNE, such as a limited ability to reveal global relationships in the data. Therefore, we are interested in extending our approach to other techniques that better address this problem, such as UMAP [27] and HSNE [32]. To conclude, we believe that our technique is an enabler for more interactive high-dimensional data analysis, in particular thanks to the possibility of optimizing embeddings directly in the browser.

[2]https://projector.tensorflow.org

[3]https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard

Acknowledgments

The authors wish to thank the Google AI team PAIR for supporting the development of the TensorFlow.js implementation. This work received funding through the STW Project 12720, VAnPIRe.

References

[1] S. J. Aarseth. *Gravitational N-Body Simulations*. Cambridge University Press, 2003. Cambridge Books Online.

[2] E.-a. D. Amir, K. L. Davis, M. D. Tadmor, E. F. Simonds, J. H. Levine, S. C. Bendall, D. K. Shenfeld, S. Krishnaswamy, G. P. Nolan, and D. Pe'er. viSNE enables visualization of high dimensional single-cell data and reveals phenotypic heterogeneity of leukemia. *Nature biotechnology*, 31(6):545–552, 2013.

[3] J. Barnes and P. Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324:446, 1986.

[4] B. Becher, A. Schlitzer, J. Chen, F. Mair, H. R. Sumatoh, K. W. W. Teng, D. Low, C. Ruedl, P. Riccardi-Castagnoli, and M. Poidinger. High-dimensional analysis of the murine myeloid cell system. *Nature immunology*, 15(12):1181–1189, 2014.

[5] H. Bezerra, E. Eisemann, X. Decoret, and J. Thollot. 3d dynamic grouping for guided stylization. In *NPAR 2008: Proceedings of the 6th International Symposium on Non-photorealistic Animation and Rendering*, pp. 89–95. ACM, 2008.

[6] S. Carter, Z. Armstrong, L. Schubert, I. Johnson, and C. Olah. Activation atlas. *Distill*, 2019. https://distill.pub/2019/activation-atlas. doi: 10.23915/distill.00015

[7] D. M. Chan, R. Rao, F. Huang, and J. F. Canny. t-sne-cuda: Gpu-accelerated t-sne and its applications to modern data. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 330–338. IEEE, 2018.

[8] Z. Chen, Z. He, X. Liu, and J. Bian. Evaluating semantic relations in neural word embeddings with biomedical and general domain knowledge bases. *BMC medical informatics and decision making*, 18(2):65, 2018.

[9] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 537–546, 2008.

[10] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pp. 577–586. ACM, 2011.

[11] J.-D. Fekete and R. Primet. Progressive analytics: A computation paradigm for exploratory data analysis. *arXiv preprint arXiv:1607.05162*, 2016.

[12] F. M. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE transactions on visualization and computer graphics*, 2018.

[13] T. Höllt, N. Pezzotti, V. van Unen, F. Koning, E. Eisemann, B. Lelieveldt, and A. Vilanova. Cytosplore: Interactive immune cell phenotyping for large single-cell datasets. In *Computer Graphics Forum*, vol. 35, pp. 171–180, 2016.

[14] T. Höllt, N. Pezzotti, V. van Unen, F. Koning, B. P. Lelieveldt, and A. Vilanova. Cyteguide: Visual guidance for hierarchical single-cell analysis. *IEEE Transactions on Visualization and Computer Graphics*, 24, 2017.

[15] S. Ingram and T. Munzner. Dimensionality reduction for documents with nearest neighbor queries. *Neurocomputing*, 150:557–569, 2015.

[16] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[17] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau. A cti v is: Visual exploration of industry-scale deep neural network models. *IEEE transactions on visualization and computer graphics*, 24(1):88–97, 2018.

[18] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pp. 462–466, 1952.

[19] M. Kim, M. Choi, S. Lee, J. Tang, H. Park, and J. Choo. Pixelsne: Visualizing fast with just enough precision via pixel-aligned stochastic neighbor embedding. *arXiv preprint arXiv:1611.02568*, 2016.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds., *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. 2012.

[21] S. Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[22] O. D. Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *Visualization Symposium (PacificVis), 2011 IEEE Pacific*, pp. 171–178, 2011.

[23] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.
[24] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
[25] N. Li, V. van Unen, T. Höllt, A. Thompson, J. van Bergen, N. Pezzotti, E. Eisemann, A. Vilanova, S. M. Chuva de Sousa Lopes, B. P. Lelieveldt, and F. Koning. Mass cytometry reveals innate lymphoid cell differentiation pathways in the human fetal intestine. *Journal of Experimental Medicine*, 2018.
[26] S. Liu, P.-T. Bremer, J. J. Thiagarajan, V. Srikumar, B. Wang, Y. Livnat, and V. Pascucci. Visual exploration of semantic relationships in neural word embeddings. *IEEE transactions on visualization and computer graphics*, 24(1):553–562, 2018.
[27] L. McInnes and J. Healy. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.
[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
[29] M. Muja and D. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.
[30] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.
[31] N. Pezzotti. High dimensional inspector, 2017.
[32] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical stochastic neighbor embedding. In *Computer Graphics Forum*, vol. 35, pp. 21–30, 2016.
[33] N. Pezzotti, T. Höllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Deepeyes: Progressive visual analytics for designing deep neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):98–108, 2018.
[34] N. Pezzotti, B. Lelieveldt, L. van der Maaten, T. Hollt, E. Eisemann, and A. Vilanova. Approximated and user steerable tsne for progressive visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2016.
[35] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pp. 832–837, 1956.
[36] C. Stolper, A. Perer, and D. Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1653–1662, 2014.
[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.
[38] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*, pp. 287–297, 2016.
[39] J. W. Tukey. The future of data analysis. *The Annals of Mathematical Statistics*, pp. 1–67, 1962.
[40] C. Turkay, N. Pezzotti, C. Binnig, H. Strobelt, B. Hammer, D. A. Keim, J.-D. Fekete, T. Palpanas, Y. Wang, and F. Rusu. Progressive data science: Potential and challenges. *arXiv preprint arXiv:1812.08032*, 2018.
[41] L. Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.
[42] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.
[43] V. van Unen, T. Hollt, N. Pezzotti, N. Li, M. J. T. Reinders, E. Eisemann, A. Vilanova, F. Koning, and B. P. F. Lelieveldt. Interactive visual analysis of mass cytometry data by hierarchical stochastic neighbor embedding reveals rare cell types. *Nature Communications*, 8, 2017.
[44] J. Venna, J. Peltonen, K. Nybo, H. Aidos, and S. Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *The Journal of Machine Learning Research*, 11:451–490, 2010.
[45] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pp. 311–321. Society for Industrial and Applied Mathematics, 1993.