Linear tSNE Optimization for the Web

Nicola Pezzotti, Alexander Mordvintsev, Thomas Höllt,
Boudewijn P.F. Lelieveldt, Elmar Eisemann and Anna Vilanova

Abstract—The t-distributed Stochastic Neighbor Embedding (tSNE) algorithm has become in recent years one of the most used and insightful techniques for the exploratory data analysis of high-dimensional data. tSNE reveals clusters of high-dimensional data points at different scales while it requires only minimal tuning of its parameters. Despite these advantages, the computational complexity of the algorithm limits its application to relatively small datasets. To address this problem, several evolutions of tSNE 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 tSNE embedding for large datasets. In this work, we present a novel approach to the minimization of the tSNE objective function that heavily relies on modern graphics hardware and has linear computational complexity. Our technique does not only beat the state of the art, but can even be executed on the client side in a browser. We propose to approximate the repulsion forces between data points using adaptive-resolution textures that are drawn at every iteration with WebGL. This approximation allows us to reformulate the tSNE minimization problem as a series of tensor operation that are computed with TensorFlow.js, a JavaScript library for scalable tensor computations.

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 [25]. In recent years, non-linear dimensionality reduction techniques, also known as manifold learning algorithms, became a powerful tools for mining knowledge from data, e.g., the presence of clusters at different scales. Manifold learning algorithms for data visualization reduce the dimensionality of the points to 2 or 3 dimensions while preserving some characteristic of the data such as the preservation of the local neighborhoods. The success of this approach is motivated by the fact that most of the real-world data satisfy the "manifold hypothesis", i.e., it lies on relatively low-dimensional manifolds embedded in a high-dimensional space. The manifolds are typically mapped to a lower dimensional space and visualized and analyzed, for example, in a scatterplot.

The t-distributed Stochastic Neighbor Embedding (tSNE) algorithm [27] has been accepted as the state of the art for nonlinear dimensionality reduction applied to visual analysis of high-dimensional space in several application areas, such as life sciences [2], [4], [15] and machine learning model understanding and human-driven supervision [9], [17], [21]. tSNE can be logically 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 KullbackLeibler divergence [12] of a similarly computed joint probability distribution that measures the closeness of the points in the low dimensional space. The memory and computational complexity of the algorithm is $O\left(N^2\right)$, where N is the number of data points.

- N. Pezzotti and A. Mordvintsev are with Google Research, Zürich, Switzerland. E-mail: {pezzotti,moralex}@google.com
- N. Pezzotti, T. Hollt, B. P.F. Lelieveldt, E. Eisemann, and A. Vilanova are with the Department of Intelligent Systems, Delft University of Technology, Delft, the Netherlands.
- B. P.F. Lelieveldt is with the Division of Image Processing, Department of Radiology, Leiden University Medical Center, Leiden, the Netherlands.

Given its popularity, research efforts have been spent on improving the computational and memory complexity of the algorithm. While many works focused on the improvement of the similarity computation [16], [19], [20], [24], [26], only limited effort have been spent in improving the minimization algorithm employed for the creation of the embedding [11], [16], [26]. The most notable of these improvements is the Barnes-Hut-SNE that makes use of an N-body simulation approach [1] to approximate the repulsive forces between the data points. Despite the improvements, the minimization requires many minutes using a highly-optimized C++ implementation.

In this work we focus on the minimization of the objective function for the creation of the embedding. We observe that the heavy tail of the t-Student distribution used by tSNE makes the application of the N-body simulation not particularly effective. To address this problem we propose a novel minimization approach that embraces this characteristic and we reformulate the gradient of the objective function as a function of scalar fields and tensor operations. Our technique has linear computational and memory complexity and, more importantly, is implemented in a GPGPU fashion. The latter allowed us to implement a version for the browser that minimizes the objective function for standard datasets in a matter of seconds.

The contribution of this work is twofold:

- A linear complexity minimization of the tSNE objective function that makes use of the modern WebGL rendering pipeline. Specifically, we
  - approximate the repulsive forces between data points by drawing low-resolution textures and
  - we adopt a tensor-based computation of the objective function's gradient.
- An efficient implementation of our result that is released as part of Google's TensorFlow.js library

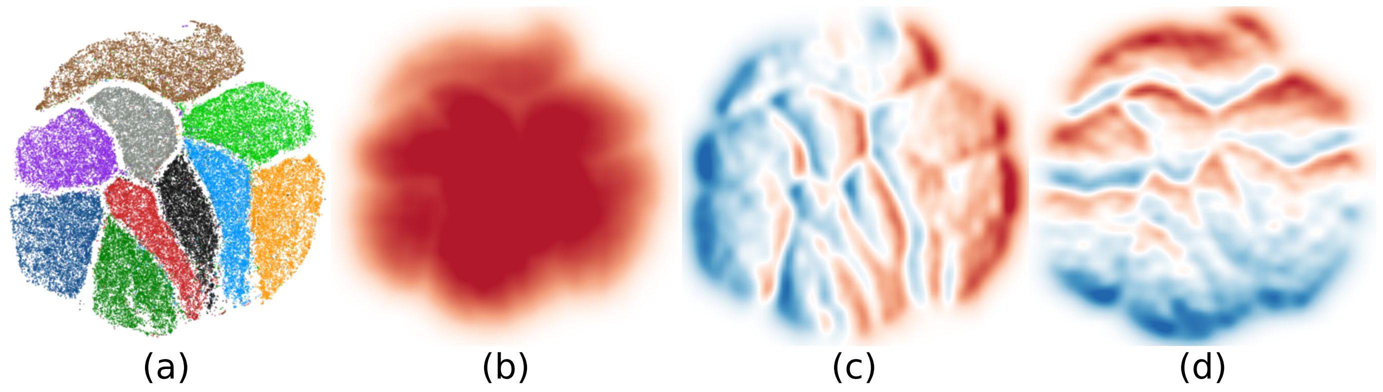

Fig. 1. **Overview** of our approach. The MNIST dataset contains images of handwritten digits and is embeedded in a 2-dimensional space (a). The minimization of the objective function is computed in linear time by making use of a scalar field (b) and a 2-dimensional vector field (c-d). The fields are computed on the GPU by splatting properly designed kernels using the additive blending function of the modern rendering pipeline. The rest of the minimization is treated as a tensor computation pipeline that is computed on the GPU using TensorFlow.js

The rest of the paper is structured as follows. In the next section, we provide a theoretical primer on the tSNE algorithm that is needed to understand the related work (Section 3) and our contributions (Section 4). In Section 5, we provide the details regarding our implementation, released within Google's TensorFlow.js library.

2 tSNE

In this section, we provide a short introduction to tSNE [27], which is needed to understand the related work and our contribution. tSNE 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 *embedding*, in the low dimensional space, in which Q faithfully represents P. This is achieved by optimizing the positions in the low-dimensional space to minimize the cost function C given by the Kullback-Leibler (KL) divergence between the joint-probability distributions P and Q:

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

Given two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ in the dataset $X = \{\mathbf{x}_1 ... \mathbf{x}_N\}$, the probability p_{ij} models the similarity of these points in the high-dimensional space. To this extent, for each point, a Gaussian kernel P_i is chosen, whose variance σ_i is defined according to the local density in the high-dimensional space and then p_{ij} is described 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 based on the local neighborhood of a data point $\mathbf{x}_i$. The perplexity value μ is a user-defined parameter that describes the effective number of neighbors considered for each data point. The value of σ_i is chosen such that for fixed μ and each i:

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

A *Student's t-Distribution* with one degree of freedom is used to compute the joint probability distribution in the low-dimensional space Q, where the positions of the data points should be optimized. Given two low-dimensional points $\mathbf{y}_i$ and $\mathbf{y}_j$, the probability q_{ij} that describes their similarity 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 gradient of the Kullback-Leibler divergence between P and Q is used to minimize C (see Eq. 1). It indicates the change in position of the low-dimensional points for each step of the gradient descent and is given by:

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

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

The gradient descent can be seen as an N*-body simulation* [1], where each data-point exerts an attractive and a repulsive force on all other points (F_i^{attr} and F_i^{rep}).

3 Related Work

We now present the work that has been done to improve the tSNE computation of tSNE embeddings in term of quality and scalability. Van der Maaten proposed the Barnes-Hut-SNE (BH-SNE) [26], 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 [29]. The minimization of the objective function is then seen as an N-body simulation, which is solved by applying the Barnes-Hut algorithm [3].

Pezzotti et al. [20] 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, they proposed the Approximated-tSNE (A-tSNE) algorithm [20], where approximated k-nearest neighborhood graphs are computed using a forest of randomized KD-trees [18]. Moreover, A-tSNE adopts the novel Progressive Visual Analytics paradigm [7], [23], allowing the user to observe the evolution of the embedding during the minimization of the objective function. This solution does not only enable a user-driven early termination of the algorithm but also led to novel discoveries in cell differentiation pathways [15].

A similar observation was later made by Tang et al. that led to the development of the LargeVis technique [24]. LargeVis uses random projection trees [5] followed by a kNN descent procedure [6] for the computation of the similarities and a different objective function that is minimized using a Stochastic Gradient Descent approach [10]. Despite the improvements, both the A-tSNE and LargeVis tools require 15 to 20 minutes to compute the embedding of the MNIST dataset [14], a 784-dimensional dataset of 60k images of handwritten digits, that is often used as benchmark for manifold-learning algorithms. Better performance is achieved by the UMAP algorithm [16], which provides a different formulation of the dimensionality-reduction problem as a cross-entropy minimization between topological representations. Computationally, UMAP follows very closely LargeVis and adopts a kNN descent procedure [6] and Stochastic Gradient Descent minimization of the objective function.

A completely different approach is taken in the Hierarchical Stochastic Neighbor Embedding algorithm (HSNE) [19]. HSNE efficiently builds a hierarchical representation of the manifolds and embeds only a subset of the initial data that represent an overview of the available manifolds. This approach can embed the MNIST dataset in less than 2 minutes. The user can “drill-in” the hierarchy by requesting more detailed embeddings that reveal smaller clusters of data points. HSNE is implemented in the Cytosplore [8] tool and led to the discovery of new cell populations [28] in large samples, i.e., containing more that 5 million cells. While HSNE produces better embeddings due to an easier minimization process, it has the downside that it does not produce a single embedding that depicts the complete dataset but it requires the user to actively explore the data and it generates embeddings on request.

The techniques presented so far do not take advantage of the target domain, in which the data is embedded. As a matter of fact, tSNE is mostly used for data visualization in 2-dimensional scatterplots, while the previously introduced techniques are general and can be used for higher dimensional spaces. Based on this observation, Kim et al. introduced the PixelSNE technique [11] that employs a N-body simulation approach similar to the BH-SNE, but quantizes the embedding space to the pixels used for visualizing the embedding. However, PixelSNE requires to scale the number of used pixels with respect to the size of the dataset in order to achieve a good embedding quality due to the quantization of the embedding space.

In this work, we take advantage of the 2-dimensional domain in which the embedding resides and we propose a more efficient way to minimize the tSNE objective function. Contrary to PixelSNE we observe that, by quantizing only the 2-dimensional space for the computation of the repulsive forces presented in Equation 8, embeddings that are hardly distinguishable from those generated by the BH-SNE implementation are computed in a fraction of the time. Moreover, we this approach allows to develop a linear complexity GPGPU implementation that runs in the client side of the browser.

4 Linear complexity tSNE minimization

In this section, we present our approach to minimize the objective function, presented in Equation 1, by rewriting its gradient, presented in Equation 7. The computation of the gradient relies on a scalar field $\mathcal{S}$ and a vector field $\mathcal{V}$ that are computed in linear time on the GPU.

4.1 Gradient of the objective function

The gradient of the objective function has the same form as the previously introduced one:

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

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. Our main contribution is to rewrite the computation of the gradient as a form of a scalar field $\mathcal{S}$ and a vector field $\mathcal{V}$.

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

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

Intuitively, $\mathcal{S}$ represents the density of the points in the embedding space, according to the t-Student distribution, and it is used to compute the normalization of the joint probability distribution Q. An example of the field $\mathcal{S}$ is shown in Figure 1b. The vector field $\mathcal{V}$ represents the directional repulsive force applied to the entire embedding space. An example of $\mathcal{V}$ is presented in Figure 1c-d, where the horizontal and vertical components are visualized separately. In the next section, we will present how both $\mathcal{S}$ and $\mathcal{V}$ are computed with a complexity of $O(N)$ and sampled in constant time. For now, we assume these fields given and we present how the gradient of the objective function are derived from these two fields, accelerating hereby their calculation drastically.

For the attractive forces, we adopt the restricted neighborhood contribution as presented in the Barnes-Hut-SNE technique [26]. The rationale of this approach is that, by imposing a fixed perplexity to the Gaussian kernel, only a limited number of neighbors effectively apply an attractive force on any given point (see Equation 3 and 4). Therefore we limit the number of contributing points to a multiple of the value of perplexity, equal to three times the value of the chosen perplexity, effectively reducing the computational and memory complexity to $O(N)$, since $k \ll N$ where k is the size of the neighborhood.

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

The normalization factor Z, as it was presented in Equation 6, has complexity $O\left(N^2\right)$. In our approach we compute $\hat{Z}$ in linear time by sampling the scalar field $\mathcal{S}$.

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

Note that Z and $\hat{Z}$ formulation is identical but, since we assume that $\mathcal{S}$ is computed in linear time, computing $\hat{Z}$ has linear complexity. Moreover, since $\hat{Z}$ does not depend on the point $\mathbf{y}_i$, for which we are computing the gradient, it needs to be computed only once for all the points.

The repulsive force assumes even a simpler form

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

being the value of the vector field $\mathcal{V}$ in the location identified by the coordinates $\mathbf{y}_i$ normalized by $\hat{Z}$. Similarly as for $\hat{Z}$, $\hat{F}^{\text{rep}}$ 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 we achieve this result by using the WebGL rendering pipeline to compute an approximation of these fields.

4.2 Computation of the fields

In the previous section, we formulated the gradient of the objective function as dependent from a scalar field $\mathcal{S}$ and a vector field $\mathcal{V}$. If the fields are evaluated independently, the complexity of the approach is $O\left(N^2\right)$ due to the summation in Equations 10 and 11. We achieve a linear complexity by precomputing and approximating the fields on the GPU using textures of appropriate resolution. An example of the fields for the MNIST dataset [14] is given in Figure 1b-d.

A similar approach is used for Kernel Density Estimation [22] that has applications in visualization [13] and non-parametric clustering [8]. In this setting, given a number of points, the goal is to estimate a 2-dimensional probability density function, from which the points were sampled. This is usually achieved by overlaying a Gaussian kernel, whose σ has to be estimated, on top of every data point.

Lampe et al. [13] were the first to propose a computation of the kernel density on the GPU for a visualization purpose. They observed that the Gaussian kernel used for estimating the density has a limited support, i.e., having value 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, instead of the points, little quads that are textured with a precomputed Gaussian kernel. By using additive blending available in OpenGL, i.e., by summing the values in every pixel, the resulting drawing corresponds to the desired density function.

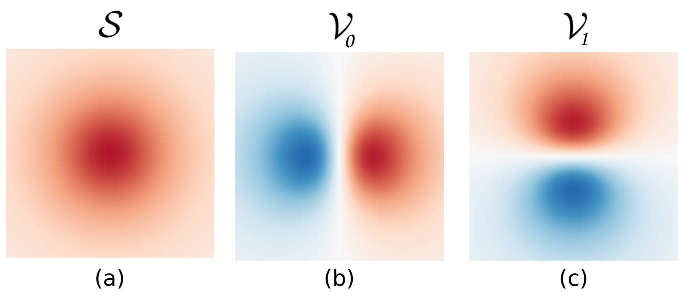

Fig. 2. **Functions** drawn over each embedding point to approximate the scalar field $\mathcal{S}$ and the 2-dimensional vector field $\mathcal{V}$.

If we analyze Equations 10 and 11, we can observe that every element in the summations for both $\mathcal{S}$ and $\mathcal{V}$ have a limited support, making it indeed very similar to the Kernel Density Estimation case discussed before. The drawn functions, however, are different and Figure 2 shows them for $\mathcal{S}$ and $\mathcal{V}$. Therefore, we can compute the fields by drawing over a texture with a single additive drawing operation. Each point is drawn as a quad and colored with a floating-point RGB texture where each channel encodes one of the functions shown in Figure 2.

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 have a fixed support in the embedding space. Therefore, given a certain embedding Y, 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 target texture according to the size of the embedding. In this way, every data point updates the value of a constant number of pixels in the target texture, effectively leading to $O(N)$ complexity for the computation of the fields.

Computing the value of $\mathcal{S}$ and $\mathcal{V}$ for a point $\mathbf{y}_i$ corresponds to extracting the interpolated value in the textures that represents the fields. This operation is extremely fast on the GPU, as WebGL natively supports the bilinear interpolation of texture values. In the next section, we provide a more detailed overview of the computational pipeline as a number of tensor operations and custom drawing operations.

5 Implementation

In this section, we present how the ideas presented in the previous section are concretely implemented in a JavaScript library that can be used to execute an efficient tSNE computation directly in the user's browser. Figure 3 shows an overview of the overall approach. We rely on TensorFlow.js, a WebGL accelerated, browser-based JavaScript library for training and deploying machine-learning models. TensorFlow.js has extensive support for tensor operations that we integrate with custom shader computations to derive the tSNE embeddings.

The randomly initialized tSNE embedding is stored in a 2-dimensional tensor. We then proceed to compute the re-

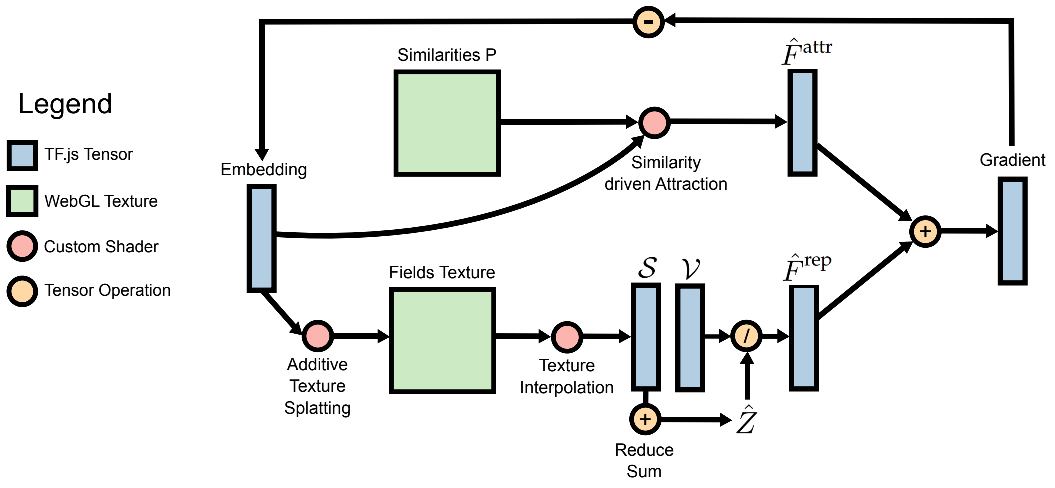

Fig. 3. **Computational worklow** of our approach. On the lower side the of the chart, the computation of the repulsive forces is presented. The fields texture is generated by the additive texture splatting presented in Section 4.2. The value of $\mathcal{S}$ and $\mathcal{V}$ are obtained through a texture interpolation and are used to compute the repulsive forces. The attractive forces are computed in a custom WebGL shader that takes as input the similarities P and the embedding. The gradient of the objective function is then computed and used to update the embedding.

pulsive forces $\hat{F}^{\text{rep}}$ and attractive forces $\hat{F}^{\text{attr}}$, shown respectively in the lower and upper side of Figure 3. The attractive forces $\hat{F}^{\text{attr}}$ are computed in a custom shader that measures the sum of the contribution of every neighboring point in the high-dimensional space. The neighborhoods are encoded in the joint probability distribution P that is stored in a WebGL texture. P can be computed server-side, for example using an approximated k-nearest-neighborhood algorithm [5], [6], [18] or by the Hierarchical-SNE technique [19]. However, we provide a WebGL implementation of the kNN-Descent algorithm [6] and the computation of P directly in the browser to enable a client-side only computational workflow.

The repulsive forces $\hat{F}^{\text{rep}}$ are computed using the approach presented in previous sections. In a custom shader, we draw for each point, whose location is defined by the value in the embedding tensor, a quad that is textured with the functions presented in Figure 2. The resulting 3-channel texture, an example of which is presented in Figure 1b-d, represents the scalar field $\mathcal{S}$ and the vector field $\mathcal{V}$. For each embedding point $\mathbf{y}i$, the values of $\mathcal{S}(\mathbf{y}i)$ and $\mathcal{V}(\mathbf{y}i)$ are stored in tensors and are computed by a custom WebGL shader that interpolates the value of the texture in the corresponding channel. The normalization factor $\hat{Z}$ is then obtained by summing all the elements in the tensor with the interpolated values of $\mathcal{S}$, an operation that is efficiently performed on the GPU by TensorFlow.js.

The remaining computational steps are computed as tensor operations. $\hat{F}^{\text{rep}}$ is obtained by dividing the interpolated values of $\mathcal{V}$ by $\hat{Z}$, and, by adding the attractive forces $\hat{F}^{\text{attr}}$, the gradient of the objective function is obtained. The gradient is then added to the embedding, hence, modifying the position of the points according to their similarities. Our work is released as part of the TensorFlow.js library and can be found on GitHub at the following address: https://github.com/tensorflow/tfjs-tsne

6 Conclusion

In this work we presented a novel approach for the optimization of the objective function of the t-Distributed Stochastic Neighbor Embedding algorithm (tSNE) that scales to large datasets in the client side of the browser. Our approach relies on modern graphics hardware to efficiently compute the gradient of the objective function from a scalar field that represents the point density and and the directional repulsive forces in the embedding space. The implementation of the technique is based on the TensorFlow.js library and can be found on GitHub at the following address: https://github.com/tensorflow/tfjs-tsne. Examples that validate our approach can also be found on GitHub https://github.com/tensorflow/tfjs-tsne-examples.

As future work, we want to perform a systematic analysis of the results of our technique, both in terms of speed and quality of the results. More specifically we plan to perform a comparison with single-embedding techniques such as LargeVis [24], UMAP [16] and PixelSNE [11]. Moreover, we want to integrate our technique in the Hierarchical-SNE technique [19] and in tools for the analysis of Deep Neural Networks such as the Embedding Projector [1], TensorBoard [2] and DeepEyes [21]. To conclude, we believe that having a scalable tSNE implementation that runs in the browsers open exciting possibilities for the development of new analytical systems.

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*, vol. 31, no. 6, pp. 545–552, 2013.
[3] J. Barnes and P. Hut, "A hierarchical o (n log n) force-calculation algorithm," *nature*, vol. 324, p. 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*, vol. 15, no. 12, pp. 1181–1189, 2014.

1. https://projector.tensorflow.org
2. https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard

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

[6] 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*. ACM, 2011, pp. 577–586.

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

[8] 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, 2016, pp. 171–180.

[9] 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*, vol. 24, no. 1, pp. 88–97, 2018.

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

[11] 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.

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

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

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] 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.

[16] L. McInnes and J. Healy, “Umap: Uniform manifold approximation and projection for dimension reduction,” *arXiv preprint arXiv:1802.03426*, 2018.

[17] 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*, vol. 518, no. 7540, pp. 529–533, 2015.

[18] M. Muja and D. Lowe, “Scalable nearest neighbor algorithms for high dimensional data,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2227–2240, 2014.

[19] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova, “Hierarchical stochastic neighbor embedding,” in *Computer Graphics Forum*, vol. 35, 2016, pp. 21–30.

[20] 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*, vol. PP, no. 99, pp. 1–1, 2016.

[21] 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*, vol. 24, no. 1, pp. 98–108, 2018.

[22] M. Rosenblatt, “Remarks on some nonparametric estimates of a density function,” *The Annals of Mathematical Statistics*, pp. 832–837, 1956.

[23] 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*, vol. 20, no. 12, pp. 1653–1662, 2014.

[24] 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*, 2016, pp. 287–297.

[25] J. W. Tukey, “The future of data analysis,” *The Annals of Mathematical Statistics*, pp. 1–67, 1962.

[26] L. Van Der Maaten, “Accelerating t-sne using tree-based algorithms,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3221–3245, 2014.

[27] L. van der Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, no. 2579-2605, p. 85, 2008.

[28] 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*, vol. 8, 2017.

[29] 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*. Society for Industrial and Applied Mathematics, 1993, pp. 311–321.

Nicola Pezzotti received his Master of Science degree from the University of Brescia, Italy, in 2011. Previously he worked as a research fellow at University of Brescia and as a research engineer in Open Technologies. He is a PhD student at Delft University of Technology in the Computer Graphics and Visualization group. He joined as visiting researcher the AVIZ/INRIA team in 2017 and Google Research in 2018. His research interests include visual analytics, machine learning and high-performance computing.

Alexander Mordvintsev received his Computer Science Master degree from Saint-Petersburg ITMO University, Russia in 2010. He worked on a number of industrial projects focusing on simulation and computer vision tasks as a research engineer in Transas Group. In 2014 he joined Google Zurich. His research interests include interpretability of machine learned models, computer vision and graphics. In 2015 he developed the DeepDream algorithm.

Boudewijn P.F. Lelieveldt received a PhD in medical image analysis from the Leiden University in 1999. He is heading the Division of Image Processing (www.lkeb.nl) at the Leiden University Medical Center, and hold a Medical Delta professor chair of Biomedical Imaging at Leiden University and Delft University of Technology. His research interest includes dimensionality reduction methods, with application in complex biomedical datasets. He is a member of IEEE.

Thomas Höllt received the Diplom (MSc) from the University of Koblenz-Landau, Germany, in 2008, and the PhD in computer science from the King Abdullah University of Science and Technology, Saudi Arabia, in 2013. He is a Postdoctoral fellow at Delft University of Technology. His research interests include visualization, computer graphics and GPGPU. He is a member of IEEE and Eurographics.

Elmar Eisemann is a professor at Delft University of Technology, heading the Computer Graphics and Visualization group. Before, he was an associate professor at Telecom ParisTech and senior researcher in the Cluster of Excellence at MPII/Saarland University. His interests include real-time and perceptual rendering, alternative representations, shadow algorithms, global illumination, and GPU acceleration techniques. In 2011, he was honored with the Eurographics Young Researcher Award.

Anna Vilanova is associate professor at the Delft University of Technology in the Computer Graphics and Visualization group. Before, she was assistant professor at the Eindhoven University of Technology. She is leading a research group in the subject of multivalued image analysis and visualization. Her research interests include visual analytics, medical visualization, volume visualization, multivalued visualization, and medical image analysis.