AtSNE: Efficient and Robust Visualization on GPU through Hierarchical Optimization

Cong Fu
The State Key Lab of CAD&CG
Zhejiang University
Hangzhou, China
fc731097343@gmail.com

Yonghui Zhang
The State Key Lab of CAD&CG
Zhejiang University
Hangzhou, China
comzyh@gmail.com

Deng Cai
The State Key Lab of CAD&CG
Alibaba-Zhejiang University Joint Institute of Frontier Technologies
dengcai@cad.zju.edu.cn

Xiang Ren
University of Southern California
Los Angeles, U.S.A.
xiangren@usc.edu

ABSTRACT

Visualization of high-dimensional data is a fundamental yet challenging problem in data mining. These visualization techniques are commonly used to reveal the patterns in the high-dimensional data, such as clusters and the similarity among clusters. Recently, some successful visualization tools (e.g., BH-t-SNE and LargeVis) have been developed. However, there are two limitations with them : (1) they cannot capture the global data structure well. Thus, their visualization results are sensitive to initialization, which may cause confusions to the data analysis. (2) They cannot scale to large-scale datasets. They are not suitable to be implemented on the GPU platform because their complex algorithm logic, high memory cost, and random memory access mode will lead to low hardware utilization. To address the aforementioned problems, we propose a novel visualization approach named as Anchor-t-SNE (AtSNE), which provides efficient GPU-based visualization solution for large-scale and high-dimensional data. Specifically, we generate a number of anchor points from the original data and regard them as the skeleton of the layout, which holds the global structure information. We propose a hierarchical optimization approach to optimize the positions of the anchor points and ordinary data points in the layout simultaneously. Our approach presents much better and robust visual effects on 11 public datasets, and achieve 5 to 28 times speed-up on different datasets, compared with the current state-of-the-art methods. In particular, we deliver a high-quality 2-D layout for a 20 million and 96-dimension dataset within 5 hours, while the current methods fail to give results due to running out of the memory.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; **Machine learning approaches**;

KEYWORDS

High-dimensional Visualization,large-scale data,GPU

ACM Reference Format:
Cong Fu, Yonghui Zhang, Deng Cai, and Xiang Ren. 2019. AtSNE: Efficient and Robust Visualization on GPU through Hierarchical Optimization. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330834

1 INTRODUCTION

Information visualization plays a key role in pattern discovery and content analysis of Big Data. The resulting knowledge we get from the visualization can benefit all walks of life and create enormous economic value. With the fast development of deep learning techniques [10], we can learn numeric representations with rich semantic information for all kinds of data. These representations are usually real vectors of hundreds or thousands of dimensions, and they are unreadable to humans. Therefore, visualizing high-dimensional and large-scale data in 2-D or 3-D space is essential, which has attracted broad research interests in data mining and visualization community [1, 11, 18]. The resulting 2-D or 3-D layouts are much easier to be understood by human and benefit many applications. For example, we can use the technique to discover the patterns in the data, we can use it for troubleshooting in training deep models, and we can use it in seeking interpretability of neural networks.

Projecting high-dimensional data with rich information into 2-D or 3-D spaces will cause unavoidable information loss. To reveal the underlying patterns in the low dimensional layout, we should preserve the intrinsic structure of the data in the original space as possible as we can. Among all kinds of linear and non-linear dimension reduction methods [1, 9, 14, 17, 18], t-SNE [11] presents the best visual effect. Later, BH-t-SNE [19] is proposed to accelerate t-SNE. However, it is still impractical for real large-scale data sets [15] because it is too time-consuming. Tang *et al.*[15] propose LargeVis to further reduce the running time, which can scale to million-point data sets. To the best of our knowledge, LargeVis is the fastest CPU visualization approach published so far.

However, we find there are two problems with the BH-t-SNE and LargeVis. Firstly, both BH-t-SNE and LargeVis cannot preserve the global structure information well. Specific manifestations include:

1) we find that the low-dimensional layouts generated by BH-t-SNE and LargeVis are very sensitive to the initialization. At projecting stage, their low-dimensional embedding of each point is initialized randomly. With different random initialization, their final layout will be very different. For instance, in Figure 1, the

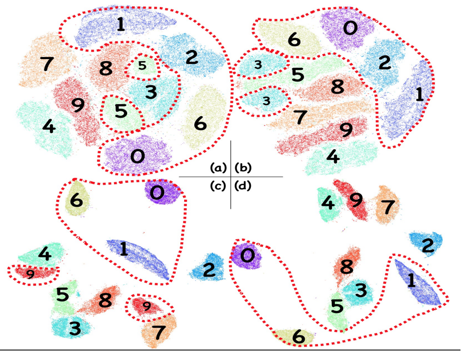

Figure 1: BH-t-SNE and LargeVis generate very different layouts under different initialization. We randomly select two seed numbers for the random number engine and run BH-t-SNE and LargeVis two times on the MNIST Dataset. Layout (a) and (b) are the outcomes of BH-tSNE, and (c) and (d) are from LargeVis. We can see that their layouts change significantly, and broken clusters emerges.

upper-row are the layouts generated by BH-t-SNE with different random initialization, and the lower-row are generated by LargeVis. In sub-figure (a), the cluster "6" lies between "2" and "0", but in sub-figure (b), the "0" cluster lies between "6" and the "2". Similar cases happen in (c) and (d) (LargeVis).

2) Broken clusters often emerge in their layouts (points with the same label form different clusters). In Figure 1, the cluster "5" in (a), "3" in (b), "9" in (c) are broken. These deformations of the layouts are not merely caused by rotation and translation transformation. This experiment indicates that BH-t-SNE and LargeVis actually cannot capture the global structure of the high-dimensional data. The reason is mainly that the two algorithms generate the low-dimensional embeddings based on K Nearest Neighbor graphs (K-NN graphs) (Please refer to Section 2 for details).

Secondly, we find BH-t-SNE and LargeVis are difficult to scale to large datasets. Million-scale datasets cannot be regarded as "large" today. LargeVis is several times faster than BH-t-SNE but still needs more than 9 hours to visualize a dataset with 4 million 100-D vectors on a 32-core CPU [15], which means we need a few days to visualize larger datasets. Moreover, LargeVis needs too much memory at the K-NN construction stage. We run LargeVis on the 20 million Amazon dataset, and it runs out of all the 128 GB memory with the default parameters and fails.

Nowadays, people usually turn to GPUs when their programs suffer from performance problems. Using GPUs for high-performance computing has achieved great success in many areas of Data Mining. Different from CPUs, GPUs are better at dealing with heavy arithmetic calculations but not very good at dealing with conditional statements in the program. To make full use of GPUs, the algorithm needs to satisfy a few requirements: 1) continuous memory access mode; 2) highly parallelizable; 3) memory-efficient (the memory of the GPU is usually much smaller than CPU's); 4) heavy arithmetic calculation tasks. We find that BH-t-SNE and LargVis are not suitable for the GPU platform. The reasons are two-fold: the low parallelizability and high memory cost (Please refre to Section 2 for details).

To address the above problems, we propose a novel approach to provide efficient and robust visualization for large-scale and high-dimensional datasets. We preserve the structure information by minimizing the distribution difference between the high-dimensional data points and the low-dimensional projected points. Further, we solve the aforementioned problems in previous works as follows:

(A) To preserve the global structure from the original space, we propose a novel hierarchical optimization method. Specifically, we first generate **anchor points** through K-means as the skeleton. Then we optimize the anchor-layout and the ordinary-point-layout simultaneously. The interactions between the anchor points and the ordinary points will transfer the global information top-down to influence the global layout, which avoids the broken clusters and presents robust layouts.

(B) Because we take the K-means centers as the anchor points, the Inverted Files (IVF) [21] can naturally be integrated in our framework to build the K-NN graph. IVF is a widely used technique in the nearest neighbor search community and suitable for GPUs, which is of low time complexity and space complexity.

Based on the characteristic of the propose method, we name it as Anchor-t-SNE (AtSNE). It is worthwhile to highlight our contributions as follows:

1. We propose a hierarchical optimization method to capture both high-level global structure and low-level local structure information, thus, it can present robust layouts.
2. We design a scalable framework which can make full use of GPUs, It enables faster and more scalable visualization on large-scale datasets.
3. Our experimental study has shown that AtSNE can achieve plausible performance on multiple public datasets. Especially on the 20 million Amazon dataset, we generate the layout with only 5 hours but the current state-of-the-art methods fail to present results.

2 RELATED WORK

In the information visualization community, many tools have been designed to visualize the geographical data, sensor data, and network data. Among them, only a few approaches can handle pure high-dimensional real vectors. Hinton *et al.*[6] proposed Stochastic Neighbor Embedding (SNE), which can generate reasonably good layouts. SNE first convert the high-dimensional Euclidean distances between data points into conditional probabilities that represent similarities. Then SNE tries to preserve the similarity among the points in the original high-dimensional space as possible as we can.

It is soon pointed out by Maaten *et al.*[11] that SNE is hard to optimize and the produced layouts often suffer from the **crowding problem**. The crowding problem is that the data points are highly crowded and there are no clear gaps between different clusters. Maaten *et al.*[11] proposed t-SNE to address the optimization problem and the crowding problem. They first balance the asymmetric conditional probability. Then they change the Gaussian assumption of low-dimensional layouts into a student's t-distribution with one degree of freedom. The main idea behind this change is that the student t-distribution repels the dissimilar points strongly, but the repulsion will not go to infinity. Consequently, the final layout will not be sensitive to the initial mapping scale and will not be highly crowded. Soon Maaten *et al.* propose BH-t-SNE to accelerate the projection by modeling the similarity among points according to a pre-built K-NN graph. The K-NN graph is constructed by a tree-based search method [19]. The BH-t-SNE can scale to larger datasets than t-SNE.

To further increase the scalability, LargeVis [15] is proposed. They formulate the visualization problem from a different perspective and propose a new probabilistic model for the graph visualization. They use the stochastic gradient descent algorithm to optimize their objective function, and the time complexity of the projection of LargeVis is $O(N)$, much less than the $O(N \log N)$ of t-SNE, where N is the size of the dataset. In their experimental study, they can perform visualization on million-scale datasets and achieve a high speed-up over t-SNE on large benchmark datasets.

Despite the success of previous works, we find some problems. (1) Both BH-t-SNE and LargeVis cannot preserve the global structure information. The low-dimensional layouts produced by their algorithms are very sensitive to the initialization, and there are broken clusters (as shown in Figure 1). This is because they all model the similarities among points based on a K-NN graph. To accelerate their methods, they choose a small K ($K \ll N$) (it is faster to build a K-NN graph with a small K. A small K also reduces the computations in the optimization). The drawback of a small K is obvious, the K-nearest neighbor graph only preserves the interactions between each point and its small local neighborhood, *i.e.*, only preserves limited local information. The problem will become worse when the data scale grows.
(2) The BH-t-SNE and LargeVis are not efficient enough on CPU and also not suitable to be implemented on GPU. Their K-NN graph construction algorithms are tree-based nearest neighbor search algorithm. They need to traverse the tree nodes to search for the nearest neighbors of a given query point, which involves many conditional statements at each tree node in the search algorithm and is hard to be parallelized. Meanwhile, tree-based nearest neighbor search algorithms lead to the random memory access mode, which is well supported on CPU but very inefficient on GPUs. In addition, there is a graph refinement step in LargeVis to improve the accuracy of the graph, which also relies heavily on the random memory access. In summary, the reason why BH-t-SNE and LargeVis cannot be extended naturally to the GPU are three-fold: (1) building K-NN graphs with tree-based methods leads to high data dependency. (2) they all rely heavily on the efficient random access mode supported by CPUs. (3) They use large amount of memory at running time.

Though BH-t-SNE is not GPU-friendly, Chan *et al.* [2] still try to accelerate it on GPUs. As discussed above, the acceleration via this naive engineering optimmization is very limited. Leland *et al.* [12] formulate this visualization problem in a different way, *i.e.*, they try to reduce the dimension via manifold learning and propose Uniform Manifold Approximation and Projection (UMAP). Specifically, they try to projecting the data by maintaining the similarity between the fuzzy topological representations from the high dimensional-space and low-dimensional space. They claim UMAP can maintain both local and global information from the high dimensional space. Meanwhile, UMAP is much faster than t-SNE. However, preserving both local and global information can be achieved only if there exists a uniform manifold for the given datasets, which may not be true for real-world datasets.

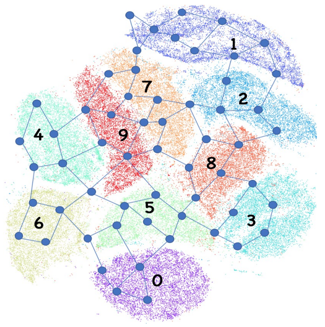

Figure 2: An illustration of our idea, visualizing with anchor points on the MNIST Dataset. The blue circles are the anchor points generated by K-means. We can preserve the global information through the interactions among the anchor points: we generate a good layout for the anchors first and project the rest nodes according to their nearby anchors. Then we refine the positions of both the anchor points and other points.

The proposed work is also closely related to graph embedding methods because we use the K-NN graph to maintain the similarity information among data points. The successful graph embedding methods such as LINE [16], node2vec [4] and DeepWalk [13] cannot handle this problem well because they are not specifically designed for the visualization. Directly using these methods to generate 2-D or 3-D embeddings for the graph may not produce meaningful layouts for data analysis.

3 ANCHOR-T-SNE

3.1 Overview

We aim to propose a visualization technique which can achieve the following goals: 1) preserve the global and local structure information from the original data space to generate high-quality low-dimensional layouts; 2) design an efficient framework for the proposed approach and make full use of GPUs to improve the scalability.

Preserve both global and local structure information. Previous approaches cannot preserve the global structure information because they model the similarity among the points through the K nearest neighbor graph (K-NN graph). The similarity among points can be interpreted as the "pulling forces" among them [15]. In the previous approaches, each point only receives the pulling forces from the nearest neighbors, which can only capture local information. We propose to introduce anchor points into the optimization to hold the global information. Figure 2 is an illustration of our idea. The anchor points (generated by K-means in our implementation) serve as the skeleton of the layout. Specifically, 1) we model the pulling forces among the anchor points to preserve the global view; 2) we make the ordinary points receive the pulling forces from the anchor points to transfer the global information in a top-down manner. 3) we model the pulling forces among the ordinary points to preserve the local structure information.

Extend the scalability with GPUs. To reduce the memory usage and increase the parallelizability, we adopt a simple but effective method to build the K-NN graph, the Inverted Files (IVF) [21]. The IVF algorithm has been widely used in text search engine on the Internet and achieved great success. It can be extended naturally to high-dimensional data. The IVF index is a set of inverted lists. The elements in each list can be retrieved efficiently according to the corresponding index key. When applied in our algorithm, the anchor points are regarded as the index keys naturally. The resulting IVF index is very small and occupies only $O(N)$ space, where N is the size of the dataset. In addition, this algorithm is very GPU-friendly.

3.2 Preliminaries

Before we present our method, we will introduce some useful notations and definitions in this section. Let X denote the set of the data points in the high-dimensional space, while Y denotes the low-dimensional counterpart of X. Let A denote the set of anchor points in the high-dimensional space, and B denotes A's low-dimensional counterpart.

Visualizing the high-dimensional data in low-dimensional space is to preserve the original structure information. One intuition is that, given two points $x_i, x_j \in X$, if x_i is close to x_j in the high-dimensional space, their low-dimensional projected counterparts, y_i and $y_j \in Y$ should also be close to each other. Further, if the similarity between x_i and x_j is subject to a certain distribution, the similarity between y_i and y_j should be subject to the same or similar distribution.

Following the prior success of t-SNE [11], we assume the similarity between the points in the high-dimensional space is subject to the Gaussian distribution, and the similarity between the points in the low-dimensional space is subject to the student's t-distribution. Let $P(n_i, n_j)$ denote the similarity between any two points n_i, n_j in the high dimensional space. We define $P(n_i, n_j)$ as:

$$\begin{aligned} P(n_i, n_j) &= \frac{P(n_j|n_i) + P(n_i|n_j)}{2} \\ P(n_j|n_i) &= \frac{exp(-||n_i - n_j||^2/2\sigma_{n_i}^2)}{Normalize(n_i)} \\ P(n_i|n_j) &= \frac{exp(-||n_i - n_j||^2/2\sigma_{n_j}^2)}{Normalize(n_j)} \end{aligned} \tag{1}$$

, where $Normalize(\cdot)$ is a normalization function.

The similarity between any two points m_i, m_j in the low-dimensional space $Q(m_i, m_j)$ is defined as:

$$Q(m_i, m_j) = \frac{(1 + ||m_i - m_j||^2)^{-1}}{Normalize(m_i)} \tag{2}$$

Let $P(X) = \{P(x_i, x_j)|x_i, x_j \in X\}$ denote the similarity distribution on X and $Q(Y) = \{Q(y_i, y_j)|y_i, y_j \in Y\}$ on Y. t-SNE proposed to preserve the structure information by minimizing the he Kullback-Leibler divergence $KL(P(X)||Q(Y))$ between $P(X)$ and $Q(Y)$. To reduce the time complexity, they do not preserve the similarity among all pair of points. Instead, they build the K-NN graph and model each similarity distribution $P(x_i), Q(y_i)$ over a small set of neighbor points of x_i. LargeVis also follows the same way. However, this leads to the problems we discussed above.

When data scale grows, only modeling the distribution over the small neighbor set will lead to severe information loss in the global view. Let $P(A)$ and $Q(B)$ denote the high and low dimensional distributions on the anchor points respectively. The anchor points are the K-means centers in our implementation, which naturally carry the global structure information. To preserve the global information, we minimize the term $KL(P(A)||Q(B))$. The local structure information can be modeled approximately over the neighbor set of given x_i. Meanwhile, to receive the global structure information, the ordinary points should also receive "pulling forces" from the anchor points. Thus, we model the similarity distribution of each $x_i \in X$ over the union of its neighbor set $N(x_i)$ and the nearest anchors $N_A(x_i)$. Let $P(X)$ denote the high-dimensional distribution on the ordinary points X with "anchor-pulling", and let $Q(Y)$ denote the low-dimensional part. The global information transferring and local information preserving can be achieved by minimizing $KL(P(X)||Q(Y))$. The loss function of AtSNE can be written as:

$$\begin{aligned} L = &\sum_i KL(P(X)||Q(Y)) + \sum_i KL(P(A)||Q(B)) \\ &+ \sum_i ||b_i - \frac{\sum_{y_k \in C_{b_i}} y_k}{|C_{b_i}|}||, b_i \in B \end{aligned} \tag{3}$$

, where C_{b_i} denotes the set of points whose K-means cluster center is b_i. The last term is a regularization term. The intuition behind is that, if a_i is the center of the cluster C_{a_i}, b_i should also be the center of C_{a_i}'s low-dimensional counterpart C_{b_i}.

The definitions of $P(X)$, $Q(Y)$, $P(A)$, and $Q(B)$ have very similar forms. Thus, we only give one detailed example here. The distribution $P(X)$ is formally defined as below:

$$\begin{aligned} P(X) &= \{P(x_i, x_j)|x_i \in X, x_j \in N(x_i) \cup N_A(x_i)\} \\ P(x_i, x_j) &= \frac{P(x_j|x_i) + P(x_i|x_j)}{2} \\ P(x_j|x_i) &= \frac{exp(-||x_i - x_j||^2/2\sigma_{x_i}^2)}{Normalize(x_i)} \\ Normalize(x_i) &= \sum_{x_k \in N(x_i) \cup N_A(x_i)} exp(-||x_i - x_k||^2/2\sigma_{x_i}^2) \end{aligned} \tag{4}$$

Similarly, we can get the definition of $Q(Y)$, $P(A)$, and $Q(B)$. Due to the space limitation, please refer to the appendix for more details.

3.3 Hierarchical Optimization Algorithm

Because there is no tractable solution to optimize Equation 3 immediately, we propose a hierarchical optimization algorithm for it. The model can be abstracted as two layers: the global optimization layer and the local optimization layer. The information flows in a top-down manner. The main idea is to optimize the two-layer layout alternatively: 1) fix the layout of the ordinary points and optimize the layout of the anchor points; 2) fix the anchor point layout and optimize the layout of ordinary points.

Specifically, the optimization pipeline is as follows:

(1) Generate the anchor points with K-means.
(2) Build Inverted Files (IVF) with the anchor points.
(3) Build a K-NN graph on ordinary points and anchor points.

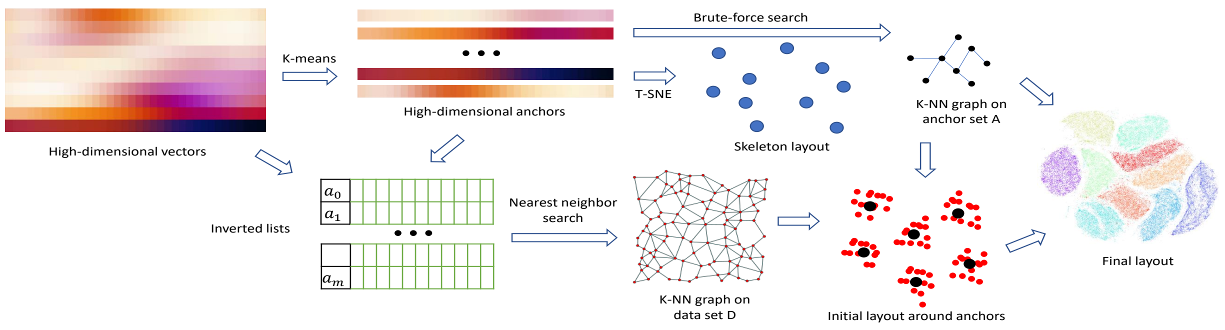

Figure 3: An illustration of the pipeline of AtSNE.

(4) In order to have better initial layout and accelerate the optimization, we first project the anchor points into the low-dimensional space until the layout is stable.
(5) Then we project the ordinary points into the low-dimensional space around the anchor point according to the cluster they belong to.
(6) Fix the coordinates of the ordinary points and optimize $KL(P(A)||Q(B))$ for one step with the SGD algorithm.
(7) Fix the coordinates of the anchor points and optimize $KL(P(X)||Q(Y))$ for one step with the SGD algorithm.
(8) Recalculate the cluster center coordinates with the ordinary points Y and replace the low-dimensional anchor coordinates with the new centers.
(9) Repeat step (5)-(7) until converge or reach the iteration limitation.

Figure 3 is an illustration of our algorithm. Integrating the IVF makes the algorithm leave small footprints in memory, based on which we adopt multiple optimization techniques to increase the efficiency and make full use of the GPU. Due to the space limitation, please refer to the Appendix for the pseudo code and implementation details. And the code is released on GitHub[1].

3.4 Complexity

The most time consuming parts of AtSNE are the K-NN graph construction and the projection.

1) The K-NN graph construction step. Let n_X be the number of points in X, d be the original dimension, and n_A be the number of anchor points in A. The time complexity of searching for the k_X nearest neighbors of one point needs about $O(n_A + n_A l)$ distance calculations and one time of ranking with the k-selection algorithm, where l is the average number of points within one inverted list. k_X is a hyper-parameter. Then the total time complexity to build the K-NN graph is $O(n_X d(n_A + n_A l) + nk_X \log n_A l)$. In the papers of BH-t-SNE and LargeVis, they did not report the time complexity of their K-NN graph construction.

2) The projection step. At each iteration, we calculate the low-dimensional similarity between each point and its neighbors, nearest anchor points and the sampled non-neighbor points. The time complexity is about $O(n_X + n_A + n_R)$, where n_R is the number of random sampled non-neighbor points (please refer to the appendix for more details). The total complexity is about $O(tn(n_X + n_A + n_R))$, where t is the number of iterations. The total complexity of updating the anchors includes re-calculating center embeddings in the low-dimensional space and performing pre-projection on A, which is about $O(n + n_A \log n_A)$. In practice, $n_X, n_A, n_R, k_X \ll n$. In conclusion, the projection complexity scales linearly with n (*i.e.*, $O(n)$), which is the same as LargeVis and faster than the $O(n \log n)$ of t-SNE [11, 15].

[1]https://github.com/zjulearning/AtSNE

4 EXPERIMENTS

4.1 Evaluation

The visual effect of the low-dimensional layout is an important evaluation of the visualization algorithms. To reduce the subjectivity as much as possible, we can evaluate the visual effect on datasets containing high-quality discriminant feature vectors in the following aspects: the points of the same label should be within the same cluster, and the borders between the clusters should be clear. The LargeVis paper [15] introduces a new evaluation named as the K-NN accuracy, which is somehow objective and quantifiable. The K-NN accuracy is obtained by testing the K-NN classifier on the low-dimensional layout and calculating the classification accuracy. It's given by

$$\begin{aligned} Acc &= \frac{\sum_i I(label(y_i), \widehat{label}(y_i))}{N}, \\ \widehat{label}(y_i) &= argmax_c \sum_{y_j \in N_D(y_i)} I(label(y_j), c) \end{aligned} \tag{5}$$

where $label(y_i)$ means the true label of point y_i and $\widehat{label}(y_i)$ means the label predicted by the K-NN classifier. $I(a, b)$ is the identity function. It equals to 1 when $a = b$. $N_D(y_i)$ is the nearest neighbor set of y_i. c is the number of categories. High K-NN accuracy intuitively reflects how much local structure information has been preserved, but it cannot show how much global structure information is preserved. The global structure information can be somehow analyzed from the visual effect. In this paper, we will evaluate AtSNE from three aspects: 1) visual effect; 2) K-NN accuracy; 3) running time. Combine the results from the three aspects and we can show the effectiveness and efficiency of AtSNE.

Table 1: We list the information of the datasets we use.

Dataset	Dimension	No. of Points	No. of Categories
CIFAR10	1024	60,000	10
CIFAR100	1024	60,000	100
MNIST	784	70,000	10
Fashion-MNIST	784	70,000	10
AG’s News	100	120,000	4
DBPedia	100	560,000	14
ImageNet	128	1,281,167	1000
Yahoo	100	1,400,000	10
Crawl	300	2,000,000	10
Amazon3M	100	3,000,000	5
Amazon20M	96	19,531,329	5

4.2 Compared Algorithm and Settings

We compare AtSNE with three popular methods, BH-t-SNE [19] (accelerated t-SNE), t-SNE-CUDA[2], and LargeVis [15]. BH-t-SNE [2], t-SNE-CUDA[3], and LargeVis [4] all release their codes on GitHub. We run them on a machine with i9-7980XE CPU and 128 GB memory. We use 32 CPU cores to test BH-t-SNE and LargeVis, while we run AtSNE and t-SNE-CUDA on a 1080Ti GPU. We do not include UMAP[12] because they only release a single-core version. It will be unfair to compare it to the others.

On comparison on different platforms. It seems unfair to compare the efficiency of algorithms immediately on different hardware platforms, but we can actually compare approximately when we take the computation power of different hardware into consideration. All the three methods are numeric-heavy tasks. Floating number instructions take up most of the time. The single-precision FLOPS (floating-point operations per second) of 32 cores on the i9-7980XE CPU is about 2.25 T (trillion). The single-precision FLOPS of 1080Ti GPU is about 11.3 T. For the same algorithm, running on 1080Ti can achieve at most 5 times speed-up than on i9-7980XE CPU. In most cases, they cannot achieve such speed-up. For algorithms which are not suitable for GPUs, it is even possible to run slower. In our experiment (Table 2), AtSNE is more than 5 times faster than LargeVis.

4.3 Parameters

In the LargeVis paper [15], they argue that it is important for a visualization tool to have default parameters, because the user may not have the time or familiar with parameter-tuning. We use the default parameters suggested by BH-t-SNE [19], t-SNE-CUDA[2], and LargeVis [15]. For our approach, we also select a group of default parameters and apply them to all the datasets. Please refer to the Appendix for more details.

4.4 Datasets

We select multiple datasets of different scales and different types (texts and images). Please see Table 1 for more details.

(1) **CIFAR10 and CIFAR100.** CIFAR10 and CIFAR100 [5] contains images of different objects of 10 categories and 100 categories respectively. We use a convolutional neural network to generate 1024-D features for each image.

[2]https://github.com/DmitryUlyanov/Multicore-TSNE
[3]https://github.com/CannyLab/tsne-cuda
[4]https://github.com/lferry007/LargeVis
[5]https://www.cs.toronto.edu/ kriz/cifar.html

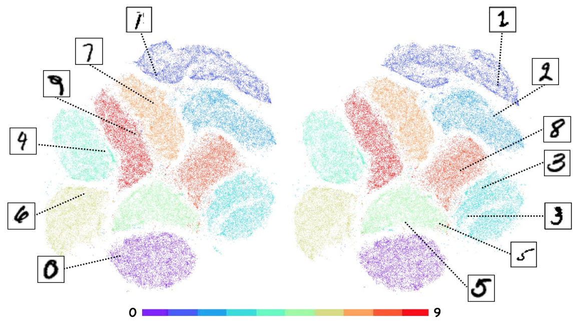

Figure 4: Two AtSNE layouts generated from different random initialization on the MNIST Dataset.

(2) **MNSIT.** MNIST[6] contains the grey images of hand-written digits from 0 to 9. The images are considered as high-dimensional vectors directly.
(3) **Fashion-MNIST.** Fashion-MNIST[7] contains 10 categories of fashion products. The images are considered as high-dimensional vectors directly.
(4) **AG’s News.** AG’s News[8] includes news of four categories (World, Sports, Business, and Science Technique). Each article is represented by a 100-D vector generated by FastText [3].
(5) **DBPedia.** DBPedia[9] consists of 560,000 Wikipedia articles with 14 categories. Each article is represented by a 100-D vector generated by FastText [3].
(6) **ImageNet.** ImageNet[10] is a dataset containing about 1000 categories of images. We extract 2048-D vectors for each image with ResNet [5]. We then project the 2048-D vectors into the 128-D space with PCA because the original dataset occupies too much memory.
(7) **Yahoo.** Yahoo[11] is the Yahoo! Answers Comprehensive Questions and Answers dataset of version 1.0. It contains the text data of 10 categories. Each article is represented by a 100-D vector generated by FastText [3].
(8) **Crawl.** The Crawl dataset[12] contains word vectors of web crawl data. The vectors are generated by FastText [3].
(9) **Amazon3M and Amazon20M.** Amazon3M and Amazon20M is obtained from the Stanford Network Analysis Project[13]. Amazon3M is sampled from the whole Amazon Reviews, which contains 3 million 100-D vectors. Amazon20M is the reviews from the book category, which contains about 20 million 96-D vectors. These vectors are generated by FastText [3].

[6]http://yann.lecun.com/exdb/mnist/
[7]https://github.com/zalandoresearch/fashion-mnist
[8]http://www.di.unipi.it/gulli/AG_corpus_of_news_articles.html
[9]https://wiki.dbpedia.org
[10]http://image-net.org/download
[11]https://webscope.sandbox.yahoo.com/
[12]http://commoncrawl.org/
[13]https://snap.stanford.edu/data/web-Amazon.html

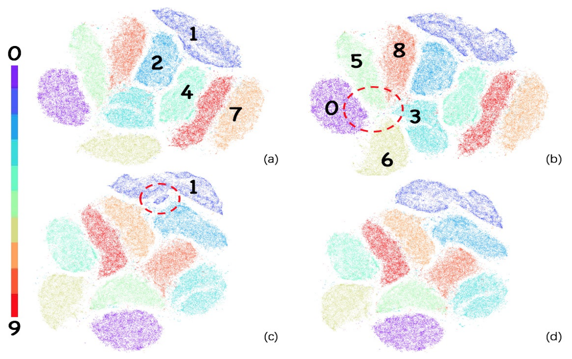

Figure 5: The result of the ablation experiments. Sub-figure (a) is generated by removing anchors from the whole optimization process. Sub-figure (b) is generated by removing anchor pre-projection and optimizing anchor points with ordinary points together. Sub-figure (c) is generated by only pre-projecting the anchor points and projecting the ordinary points around the anchors first. Then we only optimize the position of ordinary points. Sub-figure (d) is generated by the complete AtSNE.

5 RESULTS AND ANALYSIS

5.1 On Global Structure Information

Due to the space limitation, we cannot show multiple runs of all the algorithms on all the datasets. MNIST is the simplest dataset, multiple runs on which is enough to demonstrate the advantages of AtSNE. We run AtSNE with different random seed numbers, which will lead to totally different initial low-dimensional layouts. The final layouts of two different runs are shown in Figure 4. We can see that the global layouts of the two runs are almost the same, and there is only a little difference on the shape of the clusters. AtSNE captures the inter-cluster relationship well. For example, the cluster of digit "4" is close to "9" because some hand-written "4" is very similar to "9" in shape. Similarly, cluster "3" is close to "5" and "8". The result shows that AtSNE preserves the global information well. BH-t-SNE and LargeVis fail to preserve the global structure information as shown in Figure 1. The clusters shifts significantly in their layouts and some clusters broke into parts.

The results in Figure 7 are also from random runs of all the algorithm, we can see AtSNE presents the best global views on all the datasets.

5.2 Ablation Experiments

The AtSNE algorithm contains two stages: initializing layout with anchor points (initialization) and optimizing the position of anchor points and ordinary points together (refinement). The ablation experiments can be performed by removing one part each time: 1) no initialization and no refinement with anchor points; 2) no initialization but refinement with anchor points; 3) with initialization but no refinement with anchor points; 4) complete AtSNE. The results are shown in Figure 5. AtSNE without initialization and refinement is almost the same as BH-t-SNE. We can see, in sub-figure (a), the most similar cluster "1" and "7" are not very close to each other, but the dissimilar clusters "2" and "4" are very close to each other.

Table 2: The 10-NN accuracy, peak memory use, total running time of BH-t-SNE, LargeVis, t-SNE-CUDA, and AtSNE. "Speed-up" shows the speed-up times of different algorithms over LargeVis.

datasets	algorithms	10-NN Acc	time	memory (GB)	speed-up
CIFAR10	BH-t-SNE	**0.966**	5m12s	2.61	1.6
	LargeVis	0.965	8m23s	7.90	1.0
	t-SNE-CUDA	0.963	27.7s	2.17	18.1
	AtSNE	0.957	**19.6s**	**0.93**	**25.7**
CIFAR100	BH-t-SNE	0.636	9m51s	2.62	0.9
	LargeVis	0.607	8m50s	7.90	1.0
	t-SNE-CUDA	**0.646**	28.3s	2.33	18.7
	AtSNE	0.600	**19s**	**0.93**	**27.9**
MNIST	BH-t-SNE	**0.970**	5m20s	2.35	1.7
	LargeVis	0.966	8m59s	7.15	1.0
	t-SNE-CUDA	0.968	31.3s	2.33	14.7
	AtSNE	0.967	**19.6s**	**0.93**	**27.5**
FMNIST	BH-t-SNE	0.821	3m46s	2.28	2.3
	LargeVis	0.797	8m30s	7.18	1.0
	t-SNE-CUDA	**0.827**	31.1s	2.17	16.4
	AtSNE	0.822	**19.9s**	**0.93**	**25.6**
AG's News	BH-t-SNE	0.993	5m30s	0.95	1.9
	LargeVis	0.994	10m37s	2.65	1.0
	t-SNE-CUDA	0.993	39.3s	2.17	16.2
	AtSNE	**0.995**	**23s**	**0.88**	**27.7**
DBPedia	BH-t-SNE	0.993	36m8s	4.22	0.93
	LargeVis	**0.999**	33m43s	12.71	1.0
	t-SNE-CUDA	–	–	–	–
	AtSNE	**0.999**	**3m**	**2.03**	**11.2**
ImageNet	BH-t-SNE	0.412	4h7m53s	10.8	0.3
	LargeVis	0.608	1h18m45s	53.09	1.0
	t-SNE-CUDA	–	–	–	–
	AtSNE	**0.649**	**11m53s**	**4.01**	**6.6**
Yahoo	BH-t-SNE	0.537	2h17m17s	10.47	0.62
	LargeVis	0.775	1h25m17s	49.99	1.0
	t-SNE-CUDA	–	–	–	–
	AtSNE	**0.780**	**12m52s**	**4.27**	**6.6**
Crawl	BH-t-SNE	–	>24h	–	–
	LargeVis	0.688	2h34m14s	139.05	1.0
	t-SNE-CUDA	–	–	–	–
	AtSNE	**0.692**	**30m1s**	**7.19**	**5.1**
Amazon3M	BH-t-SNE	–	> 24h	–	–
	LargeVis	**0.606**	2h53m25s	104	1.0
	t-SNE-CUDA	–	–	–	–
	AtSNE	0.603	**34m4s**	**7.98**	**5.1**
Amazon20M	BH-t-SNE	–	–	–	–
	LargeVis	–	–	–	–
	t-SNE-CUDA	–	–	–	–
	AtSNE	**0.755**	**4h54m**	**19.70**	–

This indicates the global information is not preserved well. When we only add the refinement with anchors without initialization (*i.e.*, sub-figure (b)), the layout is improved a little. Cluster "0", "5", "8", "3", and "6" gather towards a common center in the red dotted circle. This behavior indicates the pulling forces from the anchor points are improving the global layout by pulling similar clusters together. However, the pulling forces are weak and they cannot push the dissimilar clusters such as "2" and "4" away from each other within 2000 iterations. In sub-figure (c), we can see the layout is much more reasonable compared with (a) and (b). Initializing the position of ordinary points near the corresponding anchor points is a good starting point, but, without pulling forces from anchor points, a part of cluster "1" is isolated out. In the sub-figure (d), the complete AtSNE generates the reasonable and well-clustered layout. This experiment shows the necessity of each part of AtSNE optimization algorithm.

5.3 K-NN accuracy, Memory and Time

Here will report the K-NN accuracy, memory-use and running time of each algorithm in Table 2. We find the K-NN accuracy of each algorithm do not change much with different K, which is

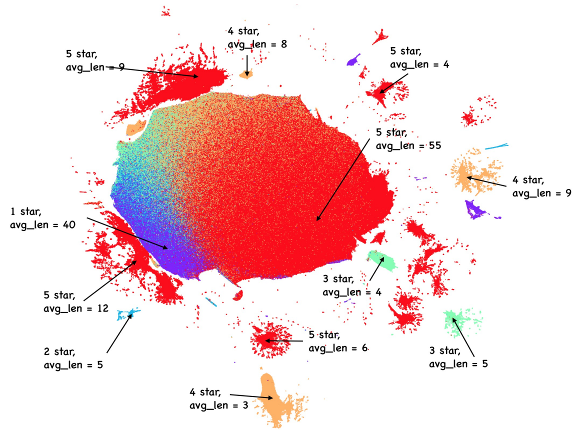

Figure 6: The visualization result of AtSNE on Amazon20M. The "stars" are the scores given by customers. "avg_len" means the average length of the comments.

consistent with the results in LargeVis [15]. Due to the limit of space, we only report the 10-NN accuracy here. We can see that the 10-NN accuracy of the three algorithms on different datasets is very similar except Yahoo and ImageNet. Though the accuracy is similar, the final layouts of different algorithms are very different, which indicates all the algorithms preserve the local information in similar degree. However, in Figure 7, AtSNE presents the best views on all the datasets, which indicates the hierarchical optimization of AtSNE can preserve both global and local information while the others cannot.

AtSNE achieves 5.1–27 times speed-up over LargeVis and 11–30 times speed-up over BH-t-SNE. Considering the difference of computational power of the hardware. Even LargeVis and BH-t-SNE can be implemented on GPU and make full use of it, they can only get about 5 times speed-up. However, they cannot be implemented on GPU efficiently because both BH-t-SNE and LargeVis consume too much memory and have low parallelizability (Section 2). On datasets larger than DBPedia, LargeVis already uses more than 12 GB memory, which cannot be placed into the memory of 1080Ti. On datasets larger than Yahoo, BH-t-SNE also uses more memory than a 1080Ti can hold. Especially, on the Crawl dataset, LargeVis uses more than 128GB memory because it uses the memory mapping techniques, *i.e.*, mapping a part of space on the disk as the extension of memory. Our memory use is many times less than them.

Both AtSNE and t-SNE-CUDA use the GPU for acceleration, but AtSNE achieves much higher speed-up (AtSNE is 40% to 70% faster than t-SNE-CUDA). This is because AtSNE is more GPU-friendly. In addition, t-SNE-CUDA suffers from memory problems. It fails on large-scale datasets due to Out-Of-Memory errors.

We did not get the results of BH-t-SNE on some datasets because it ran over 24 hours and got killed for unknown reasons. We did not get the result of LargeVis on Amazon20M because it runs out of all the memory even using the memory mapping technique.

5.4 Visual Effects

We show the visualization results of different algorithms in Figure 7. We can see that AtSNE presents the best layouts for all the datasets among the four algorithms. We do not show the effects of t-SNE-CUDA because its visual effects are the same as BH-t-SNE.

The K-NN graphs used in all the four algorithms are of very similar accuracy, which means even if they use IVF to construct the K-NN graphs, they still cannot preserve the global information as AtSNE does.

For LargeVis, there are broken clusters (*i.e.*, the points of the same label form separate clusters) on AG's News, Amazon3M, and MNIST. We can see LargeVis fails to preserve the global information and to generate stable layouts. On Fashion-MNIST, MNIST, DBPedia, ImageNet, and CIFAR10, the gaps between clusters are too wide, which indicates that LargeVis punishes distant points too much and cannot pull similar clusters to close positions.

For BH-t-SNE, there are broken clusters on AG's News, DBPedia, and MNIST, which indicates they cannot preserve the global structure information as well. In addition, it fails to generate a meaningful layout on Yahoo and ImageNet dataset. We did not present the results on Amazon3M, and Crawl datasets because BH-t-SNE is too slow. They fail to finish in 24 hours and get killed for unknown reasons.

5.5 Experiments on Amazon20M

The memory use of Amazon20M is too large for one GPU. We use product quantization (PQ) [7] to compress the original data. To further reduce the memory use, we use smaller parameters: $k_D = 50, \hat{n}_D = 100$. The rest parameters are the same as previous settings. As for the parameters of the PQ algorithm, we compress each vector into a 192-bit binary code (24 sub-quantizers, 8 bits per quantizer). We use only 5 hours and about 20 GB memory to finish the projection, but the other two algorithms cannot give the results due to the time or memory problems. The results are shown in Figure 6 and Table 2. AtSNE can get a quite good K-NN accuracy and visualization layout.

6 CONCLUSIONS

In this paper, we propose a novel method AtSNE for efficient and effective high-dimensional data visualization. It can preserve both the global and local structure information well. We design a framework for AtSNE, which can take full advantage of the GPU. In our experiments, AtSNE generates much better low-dimensional layouts than the compared algorithms, and it is the only one which can visualize the 20 million Amazon Review dataset and spends only 5 hours. The AtSNE is insensitive to different data distribution. It achieves good visualization effects on all the experimental datasets with the same default parameters, which is very user-friendly.

REFERENCES

[1] Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*. 585–591.
[2] David M Chan, Roshan Rao, Forrest Huang, and John F Canny. 2018. 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)*. IEEE, 330–338.
[3] Edouard Grave, Tomas Mikolov, Armand Joulin, and Piotr Bojanowski. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL*. 3–7.
[4] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

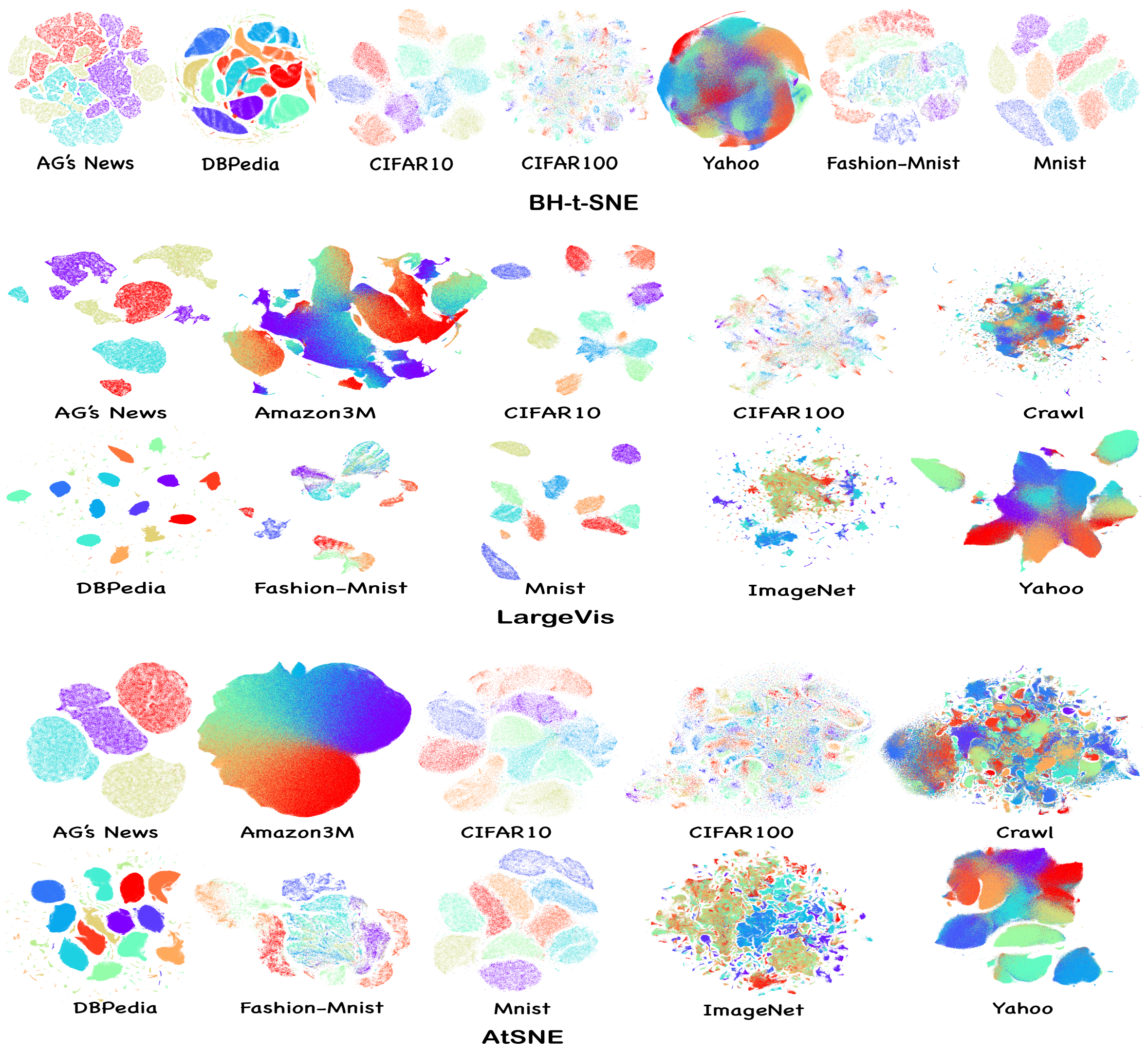

Figure 7: Visualization results of three algorithms.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[6] Geoffrey E Hinton and Sam T Roweis. 2003. Stochastic neighbor embedding. In *Advances in neural information processing systems*. 857–864.

[7] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2011), 117–128.

[8] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv:1702.08734* (2017).

[9] Ian Jolliffe. 2011. Principal component analysis. In *International encyclopedia of statistical science*. Springer, 1094–1096.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[11] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[12] Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

[13] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[14] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[15] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. 2016. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 287–297.

[16] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[17] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.

[18] Warren S Torgerson. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika* 17, 4 (1952), 401–419.

[19] Laurens Van Der Maaten. 2014. Accelerating t-SNE using tree-based algorithms. *The Journal of Machine Learning Research* 15, 1 (2014), 3221–3245.

[20] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *Proceedings of the International Conference on Very Large Data Bases* 98 (1998), 194–205.

[21] Justin Zobel and Alistair Moffat. 2006. Inverted files for text search engines. *ACM computing surveys (CSUR)* 38, 2 (2006), 6.

A OBJECTIVE FUNCTION DETAILS

We define the similarity distribution on the high-dimensional data X as:

$$\begin{aligned} P(X) &= \{P(x_i, x_j) | x_i \in X, x_j \in N(x_i) \cup N_A(x_i)\} \\ P(x_i, x_j) &= \frac{P(x_j|x_i) + P(x_i|x_j)}{2} \\ P(x_j|x_i) &= \frac{exp(-||x_i - x_j||^2/2\sigma_{x_i}^2)}{Normalize(x_i)} \\ Normalize(x_i) &= \sum_{x_k \in N(x_i) \cup N_A(x_i)} exp(-||x_i - x_k||^2/2\sigma_{x_i}^2) \end{aligned} \tag{6}$$

, where $N(x_i)$ is the nearest neighbor set of point x_i and $N_A(x_i)$ is the nearest anchor points of point x_i.

The similarity of the low-dimensional projected data Y are subjected to the student's t-distribution:

$$\begin{aligned} Q(Y) &= \{Q(y_i, y_j) | y_i \in Y, y_j \in N(y_i) \cup N_A(y_i) \cup N_R(y_i)\} \\ Q(y_j, y_i) &= \frac{(1 + ||y_i - y_j||^2)^{-1}}{Normalize(y_i)} \\ Normalize(y_i) &= \sum_{y_k \in N(y_i) \cup N_A(y_i) \cup N_R(y_i)} (1 + ||y_i - y_k||^2)^{-1} \end{aligned} \tag{7}$$

, where $N(y_i)$ is the nearest neighbor set of point y_i, $N_A(y_i)$ is the nearest anchor point set of point y_i, and $N_R(y_i)$ is the set of the points randomly sampled from the non-neighbor points of y_i. Originally, the distribution is normalized over all pair of points. To reduce the time complexity, we propose a sampling technique, which includes the above parts. Because in the t-distribution, the probability does not go to infinitesimal fast, we need the pulling forces from some distant points except from the neighbors and anchors. Randomly sampling in each iteration can approximate normalizing over all the point pairs intuitively.

We define $P(A)$ as:

$$\begin{aligned} P(A) &= \{P(a_i, a_j) | a_i \in A, a_j \in N_A(a_i)\} \\ P(a_i, a_j) &= \frac{P(a_j|a_i) + P(a_i|a_j)}{2} \\ P(a_j|a_i) &= \frac{exp(-||a_i - a_j||^2/2\sigma_{a_i}^2)}{Normalize(a_i)} \\ Normalize(a_i) &= \sum_{a_k \in N_A(a_i)} exp(-||a_i - a_k||^2/2\sigma_{a_i}^2) \end{aligned} \tag{8}$$

, where $N_A(a_i)$ is the nearest anchor points of anchor a_i.

We define $Q(B)$ as:

$$\begin{aligned} Q(B) &= \{Q(b_i, b_j) | b_i \in Y, b_j \in N_A(b_i)\} \\ Q(b_j, b_i) &= \frac{(1 + ||b_i - b_j||^2)^{-1}}{Normalize(b_i)} \\ Normalize(b_i) &= \sum_{m \neq l} (1 + ||y_i - y_k||^2)^{-1} \end{aligned} \tag{9}$$

, where $N_A(b_i)$ is the nearest anchor points of anchor b_i. Because the scale of the anchor points is small, it is fast to calculate the similarity between all the points. We do not do sampling on the anchor projection.

The loss function of AtSNE is defined as:

$$\begin{aligned} L = &\sum_i KL(P(X)||Q(Y)) + \sum_i KL(P(A)||Q(B)) \\ &+ \sum_i ||b_i - \frac{\sum_{y_k \in C_{b_i}} y_k}{|C_{b_i}|}||, b_i \in B \end{aligned} \tag{10}$$

, where C_{b_i} denotes the set of points whose K-means cluster center is b_i. The last term is a regularization term. The intuition behind is that, if a_i is the center of the cluster C_{a_i}, b_i should also be the center of C_{a_i}'s low-dimensional counterpart C_{b_i}.

B ALGORITHM PSEUDO CODE

The pseudo code of AtSNE hierarchical optimization algorithm is as below:

```
Algorithm 1 AtSNE($X$, $k_X$, $k_A$, $d_l$ $t$, $p$)
Require: High-dimensional dataset $X$, dimension $d_l$ of the low-
    dimensional layout, maximal iteration $t$, perplexity $p$.
Ensure: low-dimensional layout $Y$
 1: $n_X = |X|$, $n_A = |A|$
 2: Perform K-means on $X$ and get $A$.
 3: Build IVF index $I$ on $A$.
 4: Build K-NN graph $G_X$ on $X$ with $I$.
 5: Build K-NN graph $G_A$ on $A$ with serial-scan.
 6: Calculate the standard errors of the distributions according to $p$
 7: Random initialize $B$ as the projection of $A$.
 8: Optimize $KL(P(A)||Q(B))$ until convergence.
 9: Initialize each point in $Y$ around its cluster center in $B$.
10: for t=0:t do
11:     Fix $Y$ and optimize $KL(P(A)||Q(B))$ with one step in SGD.
12:     Fix $B$ and optimize $KL(P(X)||Q(Y))$ with one step in SGD.
13:     $B = \{\frac{\sum_{y_i \in C_{b_i}} y_i}{|C_{b_i}|} | b_i \in B\}$
14: end for
15: return $Y$
```

B.1 GPU-oriented Optimization

We adopt the following optimization to increase the efficiency and make full use of the GPU.

(1) Reorder the high-dimensional vectors according to the IVF to support continuous memory access while searching for nearest neighbors.
(2) Asynchronous data preparation to reduce waiting time of numeric calculation.
(3) We sample a subset with size limit from the whole dataset and perform K-means on it because we do not need the anchor points to be very accurate.
(4) The high-dimensional vectors occupy much memory for large-scale datasets. We can reduce the memory use with the scalar quantization [20] or the product quantization [7] techniques to compress the original data into binary codes. These techniques can also simplify the distance calculation and reduce the computational cost.
(5) We use the in-register k-selection algorithm [8] to accelerate the ranking.

C PARAMETERS

In the LargeVis paper [15], they argue that BH-t-SNE is very sensitive to parameters on different parameters and it is important for a visualization tool to have default parameters, because the user may not have the time or familiar with parameter-tuning. We follow LargeVis and compare the performance of the three methods with their default parameters.

We use the default parameters of LargeVis suggested by their paper [15]. Specifically, for LargeVis, we set the initial learning rate $\gamma = 1$, perplexity $p = 50$, $K = 150$ for K-NN graph, and the number of iteration $T = 2000$. We use the default parameters of BH-t-SNE suggested by their paper [19]. Specifically, for BH-t-SNE, we set the initial learning rate $\gamma = 200$, weight of momentum term $\theta = 0.5$, perplexity $p = 50$, and the number of iteration $T = 2000$. For t-SNE-CUDA, we use perplexity $p = 50$, learning rate $lr = 200$, and use the default setting for other parameters in the program.

For AtSNE, there are several parameters: the number of anchor points n_A, the k_D of the k-NN graph built on ordinary points, the learning rate γ, the number of iterations T, non-neighbor sampling number $\hat{n}_D$, the perplxity p, the k_A of the K-NN graph built on anchor points, and the number m of inverted lists for search. We set the default parameters for all datasets as follows: $n_A = 1000, k_D = 100, \gamma = 0.05, T = 2000, \hat{n}_D = 400, p = 50, k_A = 5, m = 50$.