International Core Journal of Engineering Volume 7 Issue 2, 2021
ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

An Overview of t-SNE Optimization Algorithms

Yifan Wang[1], Zitao Song[2], Ruicheng Zhang[3], Jiawei Shao[4], Ziheng Huang[5]

[1]School of mathematics, University of Illinois at Urbana-Champaign, Champaign, IL, 61820, USA;

[2]Xi'an Jiaotong-Liverpool University, Suzhou, JS, 215123, China;

[3]School of Software & Microelectronics, Peking University, Beijing, 102600, China;

[4]Open University of Hong Kong, Hong Kong, Hong Kong 999077, China;

[5]University of Massachusetts Amherst, 300 Massachusetts Ave, Amherst, MA 01003, USA.

Abstract

The t-distributed stochastic neighbor embedding algorithm, despite increasingly popular, are often used as a black-box data visualization method, hence making it unclear which implementation does better than others under different circumstances. This article aims to provide the reader with intuitions for choosing different t-SNE algorithms as well as the performance of different implementations of t-SNE tested under different datasets. In the work, additional optimization strategies and their analysis are also included.

Keywords

t-SNE; Optimization; Algorithm.

1. Introduction

T-SNE is one of the most widely used dimensionality reduction technique that is specialized in visualization of high-dimensional datasets. It has been widely applied into the field of computer security research, music analysis, cancer research, bio-informatics, biomedical signal processing and visualization of high-level representations learned by an artificial neural network. Many of such applications will have to deal with tremendous amount of data of various dimensions. The algorithm, however, are often used as a black-box visualizers, since different implementation of and their strengths and weaknesses are hard to understand.

In this work, a survey on different ways of optimizing t-distributed stochastic neighbor embedding(t-SNE) has been conducted with respect to the running time and quality of results. This article also aims to provide the reader with intuitions for choosing different t-SNE algorithms as well as the performance of different implementations of t-SNE under different datasets by theoretical explanations and experiments. Additionally, other optimization strategies and their analysis is also provided.

2. Backgrounds

2.1 t-distributed stochastic neighbor embedding

Suppose we have a d-dimensional dataset $\mathcal{X} = \{x_i\}_{i=1}^N \subset \mathbb{R}^d$ of size N, t-SNE is a method of finding the corresponding low dimensional embedding $\mathcal{Y} = \{y_i\}_{i=1}^N \subset \mathbb{R}^s$ with as much information preserved as possible.[1] The affinity between a pair of points(x_i, x_j) in $\mathcal{X}$ is given by the conditional probability of x_i given x_j:

ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

$$p_{i|j} = \frac{exp(-||x_i - x_j||^2/2\sigma_i{}^2)}{\sum_{k\neq i} exp(-||x_i - x_k||^2/2\sigma_i{}^2)}, \text{ with } p_{i|j} = 0 \text{ and } p_{ij} = \frac{p_{i|j} + p_{j|j}}{2N} \quad (1)$$

In the definition above, the $\sigma_i's$ are the bandwidth of the Gaussian kernels, which are given so the perplexity of P_i matches a given value. The P_i's are the conditional probability distribution of all the other points given x_i. We can also define the affinity of corresponding lower-dimensional counterparts of x_i, x_j (y_i, y_j) using t-distribution.

$$q_{ij} = \frac{(1+||y_i - y_j||^2)^{-1}}{\sum_{k\neq l}(1+||y_k - y_l||^2)^{-1}}, q_{ii} = 0 \quad (2)$$

The coordinates of the y_i's are determined by optimizing the Kullback-Leibler divergence between P and Q(the joint distributions)

$$C(\varepsilon) = KL(P||Q) = \sum_{i\neq j} p_{ij}\, log(\frac{p_{ij}}{q_{ij}}) \quad (3)$$

Note that $C(\mathcal{E})$ is not necessarily convex, so we minimize it by gradient descent method with gradient:

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

and Z is a normalizing constant

$$Z = \sum_{k\neq l}(1+\| y_k - y_l \|^2)^{-1}. \quad (5)$$

Notice that $\frac{\partial C}{\partial y_i}$ can be split into two parts in terms of the repulsive forces and attractive forces between each points:

$$\frac{1}{4}\frac{\partial C}{\partial y_i} = \sum_{j\neq i} p_{ij}\, q_{ij}Z(y_i - y_j) - \sum_{j\neq i} q_{ij}^2\, Z(y_i - y_j) = F_{\text{attr},i} - F_{\text{rep},i}. \quad (6)$$

2.2 Multiple maps t-SNE

While t-SNE is able to visualize objects as points in a low-dimensional metric map, it is limited by the metric space. These limitations prevent multi-dimensional scaling from faithfully representing non-metric similarity data, such as word associations or event co-occurrences. mm-tSNE is an extension of t-SNE. It constructs multiple mappings M to visualize non-metric pairwise similarity of the matrix, thereby alleviating the limitation of one single metric map. [2]

By the characteristics of the input similarity matrix P in higher-dimensional spaces, we normalize the original similarity matrix A so that P is symmetric, non-negative and sums up to one. The data points in two-dimensional space are also represented by q_{ij}, which is the similarity between i and j in the visualization, as the weighted sum of the pairwise similarities between the points corresponding to the input objects i and j, across all M maps. The similarity matrix among all points in the mapping m can be expressed as

$$q_{ij} = \frac{\sum_m \pi_i^{(m)}\pi_j^{(m)}(1+||y_i^{(m)} - y_j^{(m)}||^2)^{-1}}{\sum_{m'}\sum_{k\neq l}\pi_k^{(m')}\pi_l^{(m')}(1+||y_i^{(m')} - y_j^{(m')}||^2)^{-1}}, \quad (7)$$

here $y^{(m)}$ represents the low-dimensional model of object i in map m, and $\pi_i^{(m)}$ is known as the weight, it measures the importance of point i in map m.

$$\pi_i^{(m)} := \frac{e^{-\omega_i^{(m)}}}{\sum_{m'} e^{-\omega_i^{(m)}}}. \quad (8)$$

ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

Then we add a Laplacian regularization term to the cost function C(Y),

$$C(Y) = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} + \lambda \pi^T L \pi, \tag{9}$$

where $L = (diag(\sum_i p_{ij}) - P_{ij})$. And subsequently we can solve this new objective function by calculating gradient. The advantage of Laplacian regularization is that it adopts clustering structures of variables and provides more sparsity to the estimated parameters.

$$\frac{\partial C(Y)}{\partial \gamma_i^{(m)}} = 4(1-\lambda) \sum_j \frac{\partial C(Y)}{\partial d_{ij}^{(m)}} (\gamma_i^{(m)} - \gamma_j^{(m)}), \tag{10}$$

where $d_{ij}^{(m)} = ||y_i^{(m)} - y_j^{(m)}||$

$$\frac{\partial C(Y)}{\partial \pi_i^{(m)}} = \sum_j \left(\frac{2}{q_{ij} Z}(p_{ij} - q_{ij})\right) \pi_j^{(m)} (1 + d_{ij}^{(m)})^{-1} + \lambda L \pi, \tag{11}$$

where $Z = \sum_k \sum_{l \neq k} \sum_m \pi^m \pi_k^m (1 + d_{kl}^m)$.

3. Optimizations for t-SNE

3.1 LargeVis

LargeVis, published by [3], is a significant improvement over traditional t-SNE algorithm. The traditional t-SNE first compute a K-NN graph (K-nearest-neighbor graph) and then visualizes it in a low dimensional space(usually 2 to 3 dimensions). LargeVis made great improvement over the above two steps.

We first use the random projection trees to build nearest-neighbor search with the algorithm in [3].

Once the K-NN graph is constructed, we shall use principle probabilistic model for visualization of the data as presented in [3]. The idea is to preserve the similarities of the vertices in the low-dimensional space. This means that we want high continuity and high trustworthiness scores. Given a pair of vertices (v_i, v_j), we define the probability of observing a binary edge $e_{ij} = 1$ to be $P(e_{ij} = 1) = f(\| y_i - y_j \|)$, where y_i is the embedding of vertex v_i in low-dimensional space and $f(\cdot)$ is a probabilistic function of $\| y_i - y_j \|$, usually we use $f(x) = \frac{1}{1+ax^2}$ or $f(x) = \frac{1}{1+exp(x^2)}$. The comparison of different $f(\cdot)$'s can be found in [3].

We can extend $P(e_{ij} = 1)$ to $P(e_{ij}) = w_{ij}) = P(e_{ij} = 1)^{w_ij}$ and the likelihood of the graph is:

$$O = \prod_{(i,j) \in E} P(e_{ij} = 1)^{w_{ij}} \prod_{(i,j) \in \overline{E}} (1 - p(e_{ij} = 1))^{\gamma} \tag{12}$$

Taking logarithm, we have

$$O \propto \sum_{(i,j) \in E} w_{ij} \log p(e_{ij} = 1) \sum_{(i,j) \in \overline{E}} \gamma \log(1 - p(e_{ij} = 1)), \tag{13}$$

where $\overline{E}$ is the set of vertex pairs that are not observed and γ is an unified weight assigned to the negative edges.

We now need to maximize equation (12), we use negative sampling techniques [4] and rewrite the objective function as:

$$O = \sum_{(i,j) \in E} w_{ij} (\log p(e_{ij} = 1) + \sum_{k=1}^{M} E_{j_k \, Pn} (j) \gamma \log(1 - p(e_{ij} = 1))), \tag{14}$$

where M is the number of negative sample for each positive edge. Using the approach of edge sampling proposed in [5]. With this edge sampling technique, the objective function remains the same and the learning process will not be affected by the variance of the weights of the edges. Further optimization could be achieved using asynchronous stochastic gradient descent[6] resulting $O(sMN)$

ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

total run time, where s is the dimension of $\mathbb{R}^s$, N is the number of vertices and M is defined above as the number of negative samples.

3.2 Barnes-Hut t-SNE

Barnes-Hut simulation is an approximation algorithm for performing an n-body simulation, it has $O(n\log n)$ running time for a direct-sum algorithm that would have been $O(n^2)$. [7]. This approximation algorithm is useful when approximating the t-SNE gradient. As discussed in section 2.1, the gradient is split into two parts as $4(F_{attr} - F_{rep})$, each taking $O(N^2)$ to compute. Proposed by [8], we can approximate $F_{rep} = \sum_{j \neq i} q_{ij}^2 Z(y_i - y_j)$ efficiently in $O(n\log n)$ time using Barnes-Hut algorithm.

Given datapoints y_i, y_j and y_k such that $\| y_i - y_j \| \approx \| y_i - y_k \| \gg \| y_j - y_k \|$, meaning y_j contributed to F_{rep} roughly as much as y_k do. BH-algorithm [7] exploits this in three ways. First, it constructs a tree (depending on dimension of the latent space) to represent the embedding space. Then it traverse through the tree with DFS. Finally, during traversal, for all nodes in the constructed tree, decide whether the node can be used to approximate the contributions to F_{rep} of all children of that node.

For example, in a quadtree (in $\mathbb{R}^2$) where each node represents a rectangular *cell* with a particular width, height and center, every non-leaf node has 4 children dividing the cell into four sub-cells. The leaf nodes are the cells that only has one or no point from the embedding, where the root node contains all points of that embedding. For every nodes, we store the center of mass of points inside the cell, y_{cell}, that corresponds to them. The number of all points in the cell, N_{cell} are also stored in the node. Notice that it takes $O(N)$ time to construct this tree with $O(N)$ nodes.

Note that if a cell is adequately small and relatively distant from point y_i, the contributions of $-q_{ij}^2 Z(y_i - y_j)$ to F_{rep} would be approximately the same for every points y_j in that cell. So that all contributions made by the cell as a whole may be approximated by $-N_{cell} q_{i,cell}^2 Z(y_i - y_{cell})$, where $q_{i,cell} Z$ is defined to be $(1+\| y_i - y_{cell} \|^2)^{-1}$. To approximate F_{rep}, we first approximate $F_{rep} Z = -q_{ij}^2 Z^2 (y_i - y_j)$ with DFS. On each node, we determine whether it can summarize all the embedding points within a specific cell. In the process of DFS, we can also estimate $Z = \sum_{i \neq j} (1+\| y_i - y_{cell} \|^2)^{-1}$ with similar manner, so that we can now give an approximate of F_{rep} by $F_{rep} = \frac{F_{rep} Z}{Z}$.

The condition to decide whether a cell can summarize all points within it is the folowing:

$$\frac{r_{cell}}{\|y_i - y_{cell}\|^2} < \theta, \tag{15}$$

where r_{cell} is the length of diagonal of the cell and θ is a threshold value. The higher θ is the less accurate and faster the approximation is. Notice that if $\theta = 0$, resulting the algorithm to be reduced to naive algorithm. In this case, the approximation is 100% accurate but the slowest.

3.3 FIt-SNE

Fast interpolation-based t-SNE (FIt-SNE) [9] was first invented by Linderman in 2017 and was applied to visualize RNA-seq data in 2019. Generally, It interpolates the repel force in partial derivatives onto equispaced grid and is accelerated by Fast Fourier Transform (FFT), thus it has dramatically reduced the running time from $O(NlogN)$ to $O(N)$ to calculate the gradient at each iteration of gradient decent.

More specifically, suppose we want to convert $\mathcal{X}$ into s-dimension. We get the repel force of the gradients from preliminaries

$$F_{rep,k}(m) = \left(\sum_{l=1, l \neq k}^{N} \frac{y_l(m) - y_k(m)}{(1+\|y_l - y_k\|^2)^2}\right) / \left(\sum_{j=1}^{N} \sum_{l=1, l \neq j}^{N} \frac{1}{(1+\|y_l - y_k\|^2)}\right), \tag{16}$$

ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

where $y_i(j)$ denotes the jth component of y_i; $k = 1,2,\ldots,N$ and $m = 1,2,\ldots,s$. Based on Equation (16), Linderman extracts kernel $\phi(y_i) = \sum_{j=1}^{N} K(y_i, y_j)q_j$ from it. And the kernel $K(y,z)$ is either

$$K_1(y,z) = \frac{1}{(1+\|y-z\|^2)} \text{ or } K_2(y,z) = \frac{1}{(1+\|y-z\|^2)^2}. \quad (17)$$

for $y, z \in \mathbb{R}^s$. It is also easy to show that both of the kernel K_1 and K_2 are smooth functions on $\mathbb{R}^s$. After that, the critical part is to use polynomial interpolants of K to accelerate the computation.

Suppose we have $y_1,\ldots,y_M \in (y_0, y_0 + R)$ and $z_1,\ldots,z_M \in (z_0, z_0 + R)$, each interval is denoted as I_{y_0} and I_{z_0} respectively. We can formulate the interpolants of kernels K as K_p on intervals I_{y_0} and I_{z_0}

$$K_p(y,z) = \sum_{l=1}^{p}\sum_{j=1}^{p} K(\tilde{y}_j, \tilde{z}_l)L_{j,\tilde{y}}L_{l,\tilde{z}}(z), \quad (18)$$

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

$$L_{l,\tilde{y}}(y) = \prod_{j=1,j\neq l}^{p}(y - \tilde{y}_j) \,/\, \prod_{j=1,j\neq l}^{p}(\tilde{y}_l - \tilde{y}_j) \quad (19)$$

and

$$L_{l,\tilde{z}}(z) = \prod_{j=1,j\neq l}^{p}(z - \tilde{z}_j) \,/\, \prod_{j=1,j\neq l}^{p}(\tilde{z}_l - \tilde{z}_j). \quad (20)$$

p is a positive integer, $\tilde{y}_1,\ldots,\tilde{y}_p$ and $\tilde{z}_1,\ldots,\tilde{z}_p$ and a collection of $2p$ points on interval I_{y_0} and I_{z_0} and $K_p(y,z)$ is satisfying

$$K_p(\tilde{y}_j, \tilde{z}_l) = K(\tilde{y}_j, \tilde{z}_l), j, l = 1,2,\ldots,M. \quad (21)$$

Then, combined with Equation (18), the approximation of $\phi(y_i)$ can be expressed as

$$\tilde{\phi}(y_i) = \sum_{j=1}^{N} K_p(y_i, z_j)q_j. \quad (22)$$

Now, based on Equation (18) and (22) which are the polynomial interpolants on I_{y_0} and I_{z_0}, Linderman goes further to the general space $[y_{min}, y_{max}]$ by subdividing it into N_{int} intervals of equal length and each intervals will have exactly p equispaced nodes inside. And the total computational complexity to calculate them is $O(Np + (N_{int}p)log(N_{int}p))$. Since the choice of N_{int} and p is independent of the number of points N and only depends on some specific accuracy tolerance ϵ, the complexity can be reduced to $O(N)$.

Through experiment, there are some facts related to the choice of p and N_{int} worth to be mentioned. Firstly, Increasing p will reduce the number of intervals N_{int} required to reach the same level of accuracy in the computation. However, solely increasing p will lead to the raise of computation cost because with the number of points N increase and the number of intervals N_{int} decrease, computation cost will become independent of N_{int} and reduced to a function of p. Besides, it is observed that for $p \geq 3$, $N_{int}p$ remains nearly constant for a fixed performance level. Consequently, it is optimal to use $p = 3$ for all t-SNE calculations. Moreover, their simulation with $p = 3$ and $N_{int} = max\{50, \lceil y_{max} - ymin \rceil\}$ has the same performance as Barnes–Hut approximation with $(\theta = 3)$.

3.4 t-SNE CUDA

T-SNE CUDA algorithm[10] is a fully GPU-based t-SNE implementation method, that enables researchers to examine the structure of higher-dimensional data points effectively and reduce the burden of data and model calculations in modern machine learning tasks. The algorithm employs the

ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

FAISS-library, an efficient and simple GPU similarity search library. It is designed for extremely large scale data, so it's extremely helpful when it comes to mega-datasets.

Database vectors y are written as

$$y \approx q(y) = q_1(y_+ q2(y - q_1(y)) \tag{23}$$

with q_1 and q_2 being quantization functions. q_1 is a coarse quantizer, while q_2 is used to encode a more refined approximation of the residuals. Then we can construct the K-NN problem as follows:

$$\mathcal{N}_x = k - \text{argmin}_{y \in N} \|x - y_i\|, \tag{24}$$

which is an approximate asymmetric distance problem.

The algorithm first compute $\mathcal{N}_x^{IVF}$ where

$$\mathcal{N}_x^{IVF} = \tau - \text{argmin}_{\ c \in \mathcal{C}} \parallel x - c \parallel, \tag{25}$$

providing a rough approximation of coordinates of point x in terms of "centroids" in $\mathcal{C}$. Then, the nearest neighbors are construct as follows:

$$\mathcal{N} = k - \text{argmin}_{\ y \in \mathrm{N} | q_1(y) \in \mathcal{N}_x^{IVF}} \parallel x - q(y) \parallel, \tag{26}$$

By storing the reversed indices and grouping the vectors around the centroid, we can search by linear scanning a $O(\tau)$ reverse list. For implementation, select $|C| = \sqrt{N}$ and use k-Means algorithm to train the vector $\mathcal{C}$. Therefore, q_1 is the identity number of the nearest centroid. q_1 is more accurate and uses product quantization, where vector y are treated as a set of quantized sub-vectors. τ is a parameter defined by user to control the accuracy of the K-Nearest-neighbor-algorithm. As soon as the k-Nearest neighbor is calculated, a sparse matrix P_{ij}, that stores non-zero value of p_{ij}, may also be computed. Then the gravity can be effectively calculated by its decomposition into matrix operations. Let Q_{ij} be the matrix of the p_{ij} values and Y represent the $N \times 2$ matrix of the midpoint in the low dimensional space. In addition, let O be an $N \times 2$ matrix and denote $\odot$ as the Hadamard product of two matrices. We first assign the multiplication of F_{attr} giving:

$$F_{attr} = 4N y_i \sum_j p_{ij}\, q_{ij} - 4N \sum_j p_{ij}\, q_{ij} y_j, \tag{27}$$

$$F_{attr} = 4N \Big(\big(P_{ij} \odot Q_{ij}\big) O \odot Y - \big(P_{ij} \odot Q_{ij}\big) Y \Big). \tag{28}$$

Since we only need to compute $P \odot Q$ once, the equation is essentially a matrix subtraction, two matrix-matrix multiplications and two Hadamard products. We can represent P_{ij} as a sparse matrix with a non-zero value at i, j if i and j are neighborhood. When calculating matrix multiplications, we can use cuSPARSE, so that the entire algorithm runs in $O(NK)$ time.

4. Anchor-t-SNE

4.1 Overview

The Anchor-t-SNE was proposed by [11] to achieve two goals: 1) preserve the local as well as the global structure during the dimension reduction; 2) utilize full power of GPUs.

1) In previous approaches, as pointed out in , the similarity between any pairs of points can be written as attracting and repelling forces between them and "each point only receives the pulling forces from the nearest neighbors." However, this measure only represents local information [11]. Extending this idea, used K-means clustering center as anchor points, serving as a skeleton of the layout, and the global structure is preserved since the forces among anchor points can help keep the points under the skeleton in shape. At the mean time, pulling forces exerted on ordinary points can transfer the global

ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

information top-down and the pulling forces between ordinary points are modeled in order to preserve the local structure.

2) The Inverted Files[12] algorithm is used for constructing the K-NN graph, which occupies only $O(N)$ space and hence very GPU-friendly.

4.2 Preliminaries

In addition to section 2.1, we introduce $\mathcal{A}$ and $\mathcal{B}$ to be the collections of anchor points in $\mathbb{R}^d$ and its counterpart in $\mathbb{R}^s$. $\mathcal{A}$ and $\mathcal{B}$ are the K-means center(may be chosen to in other ways). Let $P(\mathcal{A})$ and $Q(\mathcal{B})$ be the distributions of $\mathcal{A}$ and $\mathcal{B}$. In order to preserve the global information, we need to minimize the $KL(P(\mathcal{A})||Q(\mathcal{B}))$ term, giving us the cost function to be:

$$C = \sum KL(P(\mathcal{X})||Q(\mathcal{Y})) + \sum KL(P(\mathcal{A})||Q(\mathcal{B})) + \sum_i \| b_i - \frac{\sum_{y_k \in C_{b_i}} y_k}{|C_{b_i}|} \|, b_i \in \mathcal{B}, \quad (29)$$

where C_{b_i} in the cluster of points whose K-means cluster center is b_i. Intuitively, if the cluster C_{a_i} has center a_i, then the cluster C_{b_i} having center b_i should be exactly the C_{a_i}'s low-dimensional counterpart.

4.3 Algorithm

provided a two-layered (global and local) hierarchical optimization algorithm for the cost function C in a top-down manner. The global layer and local layer are optimized alternatively. On one hand, we find optimized structure of the anchor points while fixing the structure of the ordinary points. On the other hand, we find optimized structure of ordinary points while fixing the structure of the anchor points. Specifically, the algorithm [11] for optimization is the following:

1 Use K-means clustering to generate the anchor points.

2 Use the anchor points to construct inverted files.

3 Build the K-NN graph with both anchor and ordinary points.

4 Project anchor points into low dimensional space until the layout is stable, so that we can make the optimization more efficient by having a better initial structure.

5 Map ordinary points into the corresponding lower-dimensional space around their respective adjacent anchor points.

6 Fix ordinary points and optimize $\mathrm{KL}(P(\mathcal{A})||Q(\mathcal{B}))$ using the SGD-algorithm.

7 Fix anchor points and optimize $\mathrm{KL}(P(\mathcal{A})||Q(\mathcal{B}))$ using the SGD-algorithm.

8 With the new coordinates of ordinary points, we replace the anchor points in lower-dimensional space by the new K-means centers.

9 Iterate from E-H until convergence.

The time complexity of AtSNE are mostly determined by two parts, the building of K-nearest-neighbor graph as well as the projection step. The first one takes $O(n_{\mathcal{X}} d(n_{\mathcal{A}} + n_{\mathcal{A}} l) + n k_{\mathcal{X}} \log n_{\mathcal{A}} l)$, where d is the dimension as in $\mathbb{R}^d$, $n_{\mathcal{X}}$ is the number of points in $\mathcal{X}$, $n_{\mathcal{A}}$ is the number of anchor points in $\mathcal{A}$. Notably, l is the average number of points within a inverted list in the IVF algorithm and $k_{\mathcal{X}}$ is a hyper-parameter. The projection step takes $O(tn(n_{\mathcal{X}} + n_{\mathcal{A}} + n_R))$, where n_R is the number of random sampled non-neighbor points specified in [11] and t is the number of iterations, so the time complexity for projection is $O(n)$. This is the same as LargeVis but AtSNE preserves much better global structure.

5. Test results

We have tested the above five different algorithms: Barnes-Hut t-SNE, LargeVis, tsne-cuda, Anchor-t-SNE, Flt-SNE by using seven benchmark dataset, which has been pre-processed by Anchor-t-SNE [11], on a server with Intel Xeon W-3175X CPU(28 Cores and 56 Threads, 32 threads used), Nvidia

ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

RTX 2080Ti GPU (for testing tsne-cuda and Anchor-t-SNE) and 128GB DDR4 2400 RAM. Detailed results are shown in the table below. Tested version of LargeVis, BH-t-SNE and TSNE-CUDA are[feb8121](https://github.com/lferry007/LargeVis/tree/feb8121e8eb9652477f7f564903d189ee663796f), [62dedde] (https:// github.com/DmitryUlyanov/Multicore-TSNE/tree/62dedde52469f3a0aeb22fdd7bce2538f17f77ef) and [efa2098] (https://github.com/CannyLab/tsne-cuda/tree/ efa209834879bba88814e74d7062539f4de07cc2) respectively. FIt-SNE was implementation from https://github.com/KlugerLab/FIt-SNE.

Table 1: Test Results on Different Datasets

Dataset	Method	10-NN accuracy	Time Taken(s)	Memory(GB)	speedup
CIFAR10	LargeVis	0.965	512	8.10	1.0
	Barnes-Hut t-SNE	0.965	320	2.65	1.6
	tsne-cuda	0.963	23	2.17	22.2
	Anchor-t-SNE	0.957	**11**	**0.93**	46.5
	FIt-SNE	**0.966**	137	\	3.7
CIFAR100	LargeVis	0.603	468	8.11	1.0
	Barnes-Hut t-SNE	0.620	501	2.65	0.9
	tsne-cuda	0.632	20	2.17	23.4
	Anchor-t-SNE	0.600	**14**	**0.94**	33.4
	FIt-SNE	**0.707**	145	\	3.4
MNIST	LargeVis	0.966	510	7.63	1.0
	Barnes-Hut t-SNE	0.971	314	2.45	1.6
	tsne-cuda	0.968	25	2.38	20.4
	Anchor-t-SNE	0.966	**16**	**0.91**	31.9
	FIt-SNE	**0.972**	158	\	3.2
Fashion-MNIST	LargeVis	0.793	497	7.50	1.0
	Barnes-Hut t-SNE	0.819	219	2.31	2.3
	tsne-cuda	0.822	25	2.30	19.9
	Anchor-t-SNE	0.820	**16**	**0.93**	31.1
	FIt-SNE	**0.866**	152	\	3.3
AG's News	LargeVis	0.994	590	2.80	1.0
	Barnes-Hut t-SNE	0.993	287	1.01	2.1
	tsne-cuda	0.994	32	2.20	18.4
	Anchor-t-SNE	0.995	**19**	**0.91**	31.1
	FIt-SNE	**0.995**	175	\	3.4
Yahoo	LargeVis	0.579	3645	56.09	1.0
	Barnes-Hut t-SNE	0.421	14375	12.74	0.3
	tsne-cuda	\	\	\	\
	Anchor-t-SNE	0.601	701	**0.91**	5.2
	FIt-SNE	**0.805**	1069	\	3.4
Amazon3M	LargeVis	0.610	7403	101.00	1.0
	Barnes-Hut t-SNE	\	\	\	\
	tsne-cuda	\	\	\	\
	Anchor-t-SNE	0.601	2050	9.01	3.6
	FIt-SNE	**0.680**	2373	\	3.1

From the table above, we note that different algorithm has divergent effects on different datasets. Using the result of LargeVis as a reference, we can conclude that Anchor-t-SNE has the fastest speed and the lowest memory usage among the five algorithms, Flt-SNE has the highest 10-NN accuracy, tsne-cuda also has excellent time and space complexity, but it may crash in the face of datasets with a larger amount of data (which is also possible for the version problem). The two CPU-based algorithms, Barnes-Hut-SNE and LargeVis, can not complete the computational task quickly, but also consume a lot of memory.

6. Additional strategies and possible further optimizations

6.1 Adaptive Learning Rate Algorithm

In the process of optimizing the target loss function, the learning rate is one of the hard-to-set hyperparameters, which has a significant impact on the visual performance of the model. The loss is usually sensitive to some directions in the parameter space, but not to other directions. Traditionally,

ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

a fixed learning rate is used to evaluate the gradient of each parameter iteration. A single learning rate is not suitable for all gradient search directions, and the loss function is usually more sensitive to certain directions in the parameter space. For example, A high learning rate will have a good effect on the direction of low curvature, but it will deviate from the direction of high curvature. Although the momentum method alleviates these problems to a certain extent, it does not solve this problem well. In this case, if it is believed that the direction sensitivity is axis-aligned to some extent, then each parameter sets a different learning rate, and it is very important to set the learning rate adaptively when optimizing the target loss function. In general, the problem can be solved by optimizing t-SNE with multiple maps [2,13], and let the leaning rate be adaptive according to the curvature. [3].

6.2 Adaptive gradient method

AdaGrad algorithm independently sets the learning rate of each model parameter in the optimization process. It adds element-wise scaling of the gradient based on the historical sum of squares in each dimension. This means that we keep a running sum of squared gradients. Then, we adjust the learning rate by dividing the learning rate by this sum.

Algorithm 2: AdaGrad(σ, Y, η)

1. **Input: constant σ, initial parameter Y, initial learning rate η**
2. $r \leftarrow 0$; (initial accumulation variables)
3. **While 1 do**
   a) $g \leftarrow \frac{\partial C(Y)}{\partial}\left(Y^{(t-1)}\right)$; (calculate gradient)
   b) $r \leftarrow r + g^o g$; (accumulate gradient)
   c) $\Delta Y \leftarrow -\frac{\eta}{\sqrt{r}+\sigma} g$; (calculate velocity updates)
   d) $Y^{(t)} = Y^{(t)} + \Delta Y$; (apply update)

6.3 RMSProp

According to Laurens[1], the learning rate in original tsne algorithm is updated after every iteration by means of the adaptive learning rate scheme described by Jacobs[14]. The algorithm in [14] can solve part of the learning rate problems, but RMSProp is better in solving Nonconvexity problems and is more likely to achieve the global minimum value.

RMSprop looks similar to the AdaGrad method. The only difference is that with RMSprop, we still retain the estimate of the square gradient, but instead of letting the estimate accumulate during training, we retain the moving average of that value.

6.4 mm t-SNE regularization based on RMSProp

The algorithm learns the loss function, so that the step size of the iteration is set adaptively during the algorithm iteration process, and by using the curvature information to update the gradient calculated in each iteration, the algorithm can reach the local optimal value faster and better during the iteration process. The time complexity of the algorithm has been reduced from $O(1/k)$ to $o(1/k^2)$.

Algorithm 3: mm t-SNE regularization(ρ, μ, σ)

1. **Input: decay rate ρ, constant σ, momentum coefficient μ**
2. $r \leftarrow 0$; (initial accumulation variables)
3. **While 1 do**
   a) $Y^{(t-1)} \leftarrow Y^{(t-1)} + \mu v^{(t-1)}$; (calculate temporary upgrade)
   b) $g \leftarrow \frac{\partial C(Y)}{\partial}\left(Y^{(t-1)}\right)$; (calculate gradient)
   c) $r \leftarrow \rho r + (1-\rho) g^o g$; (accumulate gradient)
   d) $v' = \mu v^{(t-1)} - \frac{\eta}{\sqrt{r}+\sigma} g$; (calculate velocity updates)
   e) $Y^{(t)} = Y^{(t)} + v'$; (apply update)

ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

6.5 Sampling method

[15] has proposed another way to speed up the training of t-SNE through downsampling the whole training dataset. Generally, the main idea is that by doing vanilla t-SNE only on sampled high-dimensional data, it has been shown that it indeed saves a large amount time in t-SNE's every step instead of accelerate only one part of it. Then it uses k-nearest neighborhood regression to learning how to transform the unsampled high-dimensional data to low-dimensional based on sampled data. Different sampling methods are compared with the performance of traditional random sampling.

6.6 Random walking sampling

Proposed in [16], Random walking sampling of a point is to select a starting point for the random walk and perform a length-L random walk based on the transition matrix $\widetilde{P}$. When L goes to infinity, the sampled subset are the points in the stationary distribution with highest probability.

This comes the same while sampling high-dimension data $\mathcal{X}$. When choosing one subsequent data point, the random walking algorithm is supposed to translate the distance d_{ij} between the current point point (state) x_i and remaining points (states) x_j into probabilities $\widetilde{p}_{ij}$ (Note: here $\widetilde{p}_{ij}$ has no relation with the p_{ij} in preliminaries). In other words, transition matrix $\widetilde{P}$ needs to be defined first. There are two ways to define it. One way is to use the normalized inverse of the euclidean distance of each points. That is

$$\widetilde{p}_{ij}^{1} = \frac{c}{\|x_i - x_j\| + \epsilon}, \tag{30}$$

where c is scaling constant and ϵ is used to avoid zero division.

Another way to do random walking is let $\widetilde{p}_{ij}$ exactly equal to p_{ij}, which is

$$\widetilde{p}_{ij}^{2} = p_{i|j} = \frac{exp(-||x_i - x_j||^2/2{\sigma_i}^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2{\sigma_i}^2)} \tag{31}$$

It can be observed that both $\widetilde{p}_{ij}^{1}$ and $\widetilde{p}_{ij}^{2}$ tend to be bigger while $\| x_i - x_j \|$ goes smaller. That means the algorithm is designed to walk along one cluster to another cluster. Generally, we can customise the transition matrix $\widetilde{P}$ through pre-define the metric $d(x_i, x_j)$ on space $\mathcal{X}$.

Based on that, stationary distribution π according to the transition matrix $\widetilde{P}$ can be found. Buljan states it is meaningful to sample the raw data from stationary distribution since it represents the points random walking algorithm visited in most time.

6.7 Hubness sampling

Hubness is novel concept which mainly focuses on the tendency of high-dimensional data to contain points (hubs) that frequency occur in the other points and it is first published on Tomšev's paper [17]. More specifically, the hubness of a point x_i equals to the k-occurrence $N_k(x_i)$ of point x_i, i.e., the number of x_i occur in k-nearest-neighborhood of the other points in $\mathcal{X}$. Thus, hubness seems to be a good measure of point centrality and top p percent of the raw data $\mathcal{X}$ (ranked according to hubness) is sampled deterministically.

7. Conclusion

In the work, we have initially looked at the two variants of t-SNE, namely, the t-SNE and the multiple maps t-SNE. We have then investigated algorithms that are most commonly used for optimizing t-SNE: as well as different algorithms to optimize it. Specifically, LargeVis, Barnes-Hut t-SNE, FIt-SNE, t-SNE-CUDA and Anchor-t-SNE. Finally, we have considered other strategies to improve the result of t-SNE such as adaptive learning ate algorithm and using random walk sampling method for optimizing t-SNE.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202102_7(2).0058

References

[1] Maaten, Laurens van der, and Geoffrey Hinton. 2008. "Visualizing Data Using T-Sne."

[2] Van der Maaten, Laurens, and Geoffrey Hinton. 2012. "Visualizing Non-Metric Similarities in Multiple Maps." Machine Learning 87 (1). Springer: 33–55.

[3] Shen, Xianjun, Xianchao Zhu, Xingpeng Jiang, Li Gao, Tingting He, and Xiaohua Hu. 2017. "Visualization of Non-Metric Relationships by Adaptive Learning Multiple Maps T-Sne Regularization." In 2017 Ieee International Conference on Big Data (Big Data), 3882–7. IEEE.

[4] Mikolov, Tomas, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. "Distributed Representations of Words and Phrases and Their Compositionality." ArXiv abs/1310.4546.

[5] Tang, Jian, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. 2016. "Visualizing Large-Scale and High-Dimensional Data," January. https://doi.org/10.1145/2872427.2883041.

[6] Lan, Guanghui, and Yi Zhou. 2018. "Asynchronous Decentralized Accelerated Stochastic Gradient Descent." ArXiv abs/1809.09258.

[7] Barnes, J. E., and P. Hut. 1986. "A Hierarchical O(n-Log-N) Force Calculation Algorithm." Nature 324: 446.

[8] Van Der Maaten, Laurens. 2014. "Accelerating T-Sne Using Tree-Based Algorithms." The Journal of Machine Learning Research 15 (1). JMLR. org: 3221–45.

[9] Linderman, George C., Manas Rachh, Jeremy G. Hoskins, Stefan Steinerberger, and Yuval Kluger. 2019. "Fast Interpolation-Based T-Sne for Improved Visualization of Single-Cell Rna-Seq Data." Nature Methods 16 (3). Springer Science; Business Media LLC: 243–45. https://doi.org/10.1038/s41592-018-0308-4.

[10] Chan, David M., Roshan Rao, Forrest Huang, and John F. Canny. 2018. "T-Sne-Cuda: GPU-Accelerated T-Sne and Its Applications to Modern Data." 2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 330–38.

[11] Fu, Cong, Yan Zhang, Deng Cai, and Xiang Ren. 2019. "AtSNE: Efficient and Robust Visualization on Gpu Through Hierarchical Optimization." Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.

[12] Zobel, Justin, and Alistair Moffat. 2006. "Inverted Files for Text Search Engines." ACM Comput. Surv. 38 (2). New York, NY, USA: Association for Computing Machinery: 6–es. https://doi.org/ 10.1145/ 1132956. 1132959.

[13] Xu, Weiwei, Xingpeng Jiang, Xiaohua Hu, and Guangrong Li. 2014. "Visualization of Genetic Disease-Phenotype Similarities by Multiple Maps T-Sne with Laplacian Regularization." BMC Medical Genomics 7 (2). BioMed Central: 1–9.

[14] Jacobs, Robert A. 1988. "Increased Rates of Convergence Through Learning Rate Adaptation." Neural Networks 1 (4). Elsevier: 295–307.

[15] Buljan, Matej. 2019. "Optimizing T-Sne Using Random Sampling Techniques (Dissertation)." Linnaeus University.

[16] Basirian, Saeed, and Alexander Jung. 2017. "Random Walk Sampling for Big Data over Networks." International Conference on Sampling Theory and Applications (SampTA). IEEE, 427–31.

[17] Tomasev, Nenad, Milos Radovanovic, Dunja Mladenic, and Mirjana Ivanovic. 2014. "The Role of Hubness in Clustering High-Dimensional Data." IEEE Transactions on Knowledge and Data Engineering 26 (3): 739–51.