PixelSNE: Pixel-Aligned Stochastic Neighbor Embedding for Efficient 2D Visualization with Screen-Resolution Precision

Minjeong Kim\(^1\), Minsuk Choi\(^1\), Sunwoong Lee\(^1\), Jian Tang\(^3,4\), Haesun Park\(^5\) and Jaegul Choo\(^1,2\)

\(^1\)Korea University, Korea
\(^2\)Clova AI Research, NAVER, Korea
\(^3\)Montreal Institute for Learning Algorithm, Canada
\(^4\)HEC Montreal, Canada
\(^5\)Georgia Institute of Technology, USA

Abstract

Embedding and visualizing large-scale high-dimensional data in a two-dimensional space is an important problem, because such visualization can reveal deep insights of complex data. However, most of the existing embedding approaches run on an excessively high precision, even when users want to obtain a brief insight from a visualization of large-scale datasets, ignoring the fact that in the end, the outputs are embedded onto a fixed-range pixel-based screen space. Motivated by this observation and directly considering the properties of screen space in an embedding algorithm, we propose Pixel-Aligned Stochastic Neighbor Embedding (PixelSNE), a highly efficient screen resolution-driven 2D embedding method which accelerates Barnes-Hut tree-based t-distributed stochastic neighbor embedding (BH-SNE), which is known to be a state-of-the-art 2D embedding method. Our experimental results show a significantly faster running time for PixelSNE compared to BH-SNE for various datasets while maintaining comparable embedding quality.

1. Introduction

As the importance of extracting knowledge from large-scale datasets increases, understanding and obtaining insights from them have become a critical issue in machine learning and data mining applications. In this sense, visualizing high-dimensional data in a two-dimensional (2D) or three-dimensional (3D) scatterplot is an effective approach to interpreting the overall structure of a dataset in terms of its cluster structure, potential outliers, etc. Through such visualization, one can obtain the idea about the relationships among clusters as well as those between individual data, i.e., similar or outlying clusters and/or data items [PTT\(^{12}\), SMT\(^{13}\)].

To generate a scatterplot given high-dimensional data, one can apply various dimension reduction or low-dimensional embedding approaches including traditional methods (e.g., principal component analysis [Jo02] and multidimensional scaling [Kru64a, Kru64b]) and recent manifold learning methods (e.g., isometric feature mapping [TSL00], locally linear embedding [SR03], and Laplacian eigenmaps [BN03]).

However, these methods are often not successful at preserving the high-dimensional data structure when reducing its dimension to two or three, so in response, an advanced dimension reduction technique called t-distributed stochastic neighbor embedding (t-SNE) [vdMH08] has been proposed, and it shows outstanding advantages over other methods in generating 2D scatterplots. A drawback of t-SNE is the significant computing time it requires for a large number of data items; t-SNE has a computational complexity of \(O(n^3)\), where \(n\) represents the number of data items. Although various approximation techniques attempting to accelerate the t-SNE algorithm have been proposed, such as Barnes-Hut SNE (BH-SNE) [VDM14] with a complexity of \(O(n \log n)\), it still takes a long time to apply them to large-scale data.

To tackle this issue, this paper proposes a novel framework that can significantly accelerate the 2D embedding algorithms in visualization applications. The proposed framework is motivated by the fact that most embedding approaches compute the low-dimensional coordinates with an excessively higher precision than the resolution of a typical screen space can initially handle. The screen space on which the low-dimensional embedding outputs are embedded has two distinctive properties: (1) it has a fixed embedding space range, and (2) it consists of pixels as the smallest units.

Leveraging the idea of applying just enough precision for a screen to the above mentioned state-of-the-art BH-SNE method, we propose a significantly faster alternative called pixel-aligned SNE (PixelSNE). PixelSNE is 4.8x fold faster than BH-SNE when visualizing 421,161 data items of News Aggregator dataset. To achieve this, PixelSNE adopts a limited-precision Barnes-Hut quadtree algorithm, which is a pixel-aligned quadtree (P-Quadtree) that reduces the computational complex-
ity to $O(n \times \max_{d\in\{1,2\}} \left[\log_2 r_d\right])$, where the $r_d$’s are user-controlled precision parameters along the $x$ and the $y$-axes, respectively.

Our contributions can be summarized as follows:

1. A novel framework of a highly-efficient 2D embedding approach called PixelSNE which utilizes our idea of a pixel-based precision.
2. Quantitative and qualitative analyses on various real-world datasets, which demonstrate the significant speedup PixelSNE achieves over BH-SNE and its comparable visualization quality.

This paper is organized as follows: Firstly, we review previous research. Then, we briefly introduce BH-SNE, and we describe the mathematical and algorithmic details of PixelSNE, which further accelerates BH-SNE. Lastly, we present our experimental results, followed by concluding remarks.

2. Related Work

Dimension reduction or low-dimensional embedding of data in a high-dimensional space [VdMPvdH09] has long been an active research area.

Widely-used dimension reduction methods in visualization applications include principal component analysis (PCA) [Jo02], multidimensional scaling [Kru64a, Kru64b], generative topographic mapping [BSW98], and self-organizing map [Koh61]. While these traditional methods generally focus on preserving global relationships rather than local ones, a class of nonlinear, local dimension reduction techniques called manifold learning [LV07] have been actively studied, trying to recover an intrinsic curvilinear manifold out of given high-dimensional data. Representative methods include isometric feature mapping [TLS00], locally linear embedding [SR03], Sammon mapping [Sam69], Laplacian eigenmaps [BN03], least square projection [PNML08], maximum variance unfolding [WS06], and autoencoder [HS06].

t-distributed stochastic neighbor embedding (t-SNE) [vdMH08] is a recent method that specifically focuses on visualization applications. This method is built upon stochastic neighbor embedding [HR02] and has shown its superiority in generating the 2D scatterplots capturing the local structure of high-dimensional data while also preserving the global structure. It reveals meaningful insights about data, such as clusters and outliers [KKP17, CLRP13] and is widely used to visualize high-dimensional data in various areas [ADT+13, FCS+13]. Subsequently, numerous approaches have been proposed to improve visualization quality and related performances in the 2D embedding results. For example, a neural network has been integrated with t-SNE to learn the parametric representation of 2D embedding [Maa09]. Rather than the Euclidean distance or its derived similarity information, other information types, such as relative ordering information about pairwise distances in the form of similarity triplets [vdMW12] and non-metric similarities [vdMH12], have been considered as the target information to preserve. Additionally, various other optimization criteria and their optimization approaches, such as elastic embedding [CP10] and NeRV [VPN+10], have also been proposed.

The computational efficiency and scalability of 2D embedding approaches have also been widely studied. An accelerated t-SNE based on tree-based Barnes-Hut algorithm has been proposed [VDM14]. The tree algorithm approximates the gradient which is used to optimize t-SNE embedding. Gisbrecht et al. proposed a linear approximation of t-SNE [GMH12], which extends t-SNE to a parametric mapping by offering a nonlinear function for parameter choices. More recently, an approximate but user-steerable t-SNE has been studied [PLvdM*16], which allows a user to control the degree of approximation when computing the high-dimensional similarities of data items located in user-specified areas. Inspired by this, Pezzoti et al. [PHL+16] introduced a hierarchical version of SNE, which reveals data structures at different scales by using a hierarchical representation of a dataset, while improving the computation efficiency. Additionally, a scalable 2D embedding technique called LargeVis [TLZM16] introduces an approximate K-nearest neighbor graph and an asynchronous stochastic gradient-descent, which significantly reduces the computing time with a linear time complexity in terms of the number of data items. In addition, Yang et al. presented a scalable neighbor embedding method using a genetic approximated optimization technique [YPK13].

Despite the plethora of 2D embedding approaches, to the best of our knowledge, none of the previous studies have directly exploited the low precision of screen space for developing highly efficient 2D embedding algorithms. Our novel framework of controlling the precision in return for algorithm efficiency in the proposed PixelSNE, which significantly improves the efficiency of BH-SNE, is one such example.

3. Barnes-Hut-SNE (BH-SNE)

t-SNE, the previous work of Barnes-Hut-SNE, embeds high-dimensional data into a low-dimensional space by minimizing the differences between the joint probability distributions representing pairwise relationships between high-dimensional data items $X = \{x_i \in \mathbb{R}^D\}_{i=1,\ldots,N}$, where $N$ is the number of data items and $D$ is the original dimension and low-dimensional embeddings $Y = \{y_i \in \mathbb{R}^2\}_{j=1,\ldots,N}$. In detail, t-SNE computes the Euclidean pairwise distance matrix $D_X \in \mathbb{R}^{N \times N}$ of $X$ and then converts it to the high-dimensional joint probability matrix $P \in \mathbb{R}^{N \times N}$ using a Gaussian kernel. To be specific, the probability density under a Gaussian kernel for each high-dimensional data point $x_i$ measures the similarity between $x_i$ and other points, where a bandwidth of the Gaussian kernel is pre-determined by a perplexity value which is interpreted as a smooth measure of the effective number of neighborhood points in a high-dimensional space.

Given randomly initialized $Y$, t-SNE computes the Euclidean pairwise distance matrix $D_Y \in \mathbb{R}^{N \times N}$ and then converts it to a joint probability matrix $Q \in \mathbb{R}^{N \times N}$ in low-dimensional space using a student’s $t$-distribution with one degree of freedom. Specifically, the $(i, j)$-th component $q_{ij}$ of $Q$, which represents the similarity between $y_i$ and $y_j$ in a probabilistic sense, is computed as

$$q_{ij} = \frac{1 + ||y_i - y_j||^2_2}{Z}$$

where $Z = \sum_{ij} \left(1 + ||y_k - y_l||^2_2\right)^{-1}$ is the normalization term.
t-SNE attempts to make \( Q \) as close as possible to \( P \) by minimizing the Kullback-Leibler divergence between them, i.e.,

\[
C = KL(P \| Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}
\]

It iteratively performs gradient-descent updates on \( \mathcal{Y} \) where the gradient with respect to \( y_i \) is computed as

\[
\frac{\partial C}{\partial y_i} = 4 (F_{\text{attr}} + F_{\text{rep}}) = 4 \left( \sum_{j \neq i} p_{ij} q_{ij} (y_i - y_j) \right) - \sum_{j \neq i} (q_{ij})^2 Z (y_i - y_j),
\]

where \( F_{\text{attr}} \) is the total sum of all the attractive forces, while \( F_{\text{rep}} \) denotes the sum of all the repulsive forces.

While t-SNE has a computational complexity of \( \mathcal{O}(N^2) \) by considering all the pairwise relationships, BH-SNE adopts two tree-based approximation methods to reduce this complexity. The first method, named the vantage-point tree [Yia93] approximately computes \( D_X \) and \( P \) in a sparse matrix form by approximating a pairwise distance of non-neighbor points as zeros (Fig. 1(a)), where the neighbor or non-neighbor points are determined by the given perplexity. This approximation makes it efficient to compute \( F_{\text{attr}} \); one can sum over all non-zero terms of the sparse matrix \( P \). When optimizing low-dimensional coordinates \( \mathcal{Y} \) (Fig. 1(b)), BH-SNE adopts the Barnes-Hut algorithm [BH86] to compute \( F_{\text{rep}} \) in Eq. (3). The fundamental concept is that when \( ||y_i - y_j|| \approx ||y_i - y_k|| \gg ||y_j - y_k|| \), the force of \( y_j \) on \( y_i \) is similar to that of \( y_k \) on \( y_i \) when computing the gradient. For example, if we set the representative data point of \( y_j \) and \( y_k \) as \( y_i \), the low-dimensional joint probability \( q_{ik} \) can be used to replace both \( q_{ij} \) and \( q_{ik} \). Another justification for this approximation is that the points \( y_j \) and \( y_k \) that are far from \( y_i \) would have a small influence on \( y_i \). Thus, even if both points are approximately represented as \( y_i \), the gradient-descent step remains stable. Based on this approximation, BH-SNE utilizes Barnes-Hut algorithm to find the single representative point of multiple data points.

To obtain the representative points, BH-SNE constructs a quadtree at each iteration. Given \( y_i^{(t)} \) at iteration \( t \), BH-SNE starts by forming a root node \( c_{\text{root}} \) which contains all the \( y_j^{(t)} \)'s, with the boundary of

\[
b^{(c_{\text{root}})}_{\text{min},d} = \min y_j^{(t)} \quad \text{and} \quad b^{(c_{\text{root}})}_{\text{max},d} = \max y_j^{(t)} + \epsilon, \forall d \in \{1, 2\},
\]

where \( \epsilon \) is a small constant. BH-SNE then recursively splits the cell \( c \) into four (equally-sized) quadrant cells located at “northwest,” “northeast,” “southwest,” and “southeast” by setting their boundaries accordingly. It then assigns \( y_j^{(t)} \)’s to these child cells based on the boundary information and computes their center of mass \( y_c \), i.e., the centroid of the points contained in the cell \( c \). The quadtree grows until each leaf node contains at most a single \( y_i^{(t)} \).

Next, BH-SNE traverses the quadtree via a depth-first search to determine whether \( y_i \) can represent the points contained in \( c \) based on the criterion

\[
d_c / ||y_i - y_c||_2 < \theta,
\]

where \( d_c \) represents the diagonal length of the region of \( c \) and \( \theta \) is a threshold parameter. Note that this process is repeated every iteration as the range of the coordinates of \( y_i^{(t)} \) at iteration \( t \) changes due to the gradient descent update.

4. Pixel-Aligned SNE (PixelSNE)

In this section, we present PixelSNE. Similar to BH-SNE, PixelSNE approximates the gradient of low-dimensional coordinates \( z_i \) by substituting multiple data items with a single representative point when multiple data points are located close to each other and far away from the point \( z_i \). To this end, PixelSNE introduces a limited-precision to the Barnes-Hut tree algorithm, which significantly accelerates the entire gradient-descent update process of BH-SNE. The process of PixelSNE is described in Algorithm 1; the code for PixelSNE is available at https://github.com/awesome-davian/pixelsne.

---

**Figure 1:** Overview of the proposed PixelSNE in comparison with the BH-SNE.
4.1. Pixel-Aligned Barnes-Hut Tree

A major novelty of PixelSNE is that it directly considers the screen-space coordinates \( z_i \) instead of \( y_i \). That is, given a screen resolution \( r_1 \times r_2 \), where the positive integers \( r_1 \) and \( r_2 \) represent the \( x \) and \( y \)-axis resolutions, respectively, PixelSNE embeds high-dimensional data into low-dimensional screen coordinates \( z_i \) with a range from zero to the given screen resolution:

\[
Z = \left\{ z_i : z_i = \left[ \frac{z_{i,1}}{z_{i,2}} \right] \in \mathbb{R}^2, \right\}_{\substack{1 \leq i \leq N}} \quad \text{where} \quad 0 \leq z_{i,d} < r_d, \; \forall d \in \{1,2\}.
\]

(7)

This enables PixelSNE to utilize the following property of \( z_i \) during the optimization process. The range of \( z_{i,d} \) remains fixed as \([0, r_d]\) for \( d = 1,2 \) throughout algorithm iterations, because a screen has a fixed screen resolution (Eq. (8)). Based on this newly defined screen space, we propose our modified quadtree algorithm—a pixel-aligned quadtree (P-Quadtree). In contrast to the original quadtree algorithm adopted by BH-SNE, which we refer to as a data-driven quadtree (D-Quadtree), our proposed method has several distinctive characteristics.

**One-time tree construction.** The above-mentioned property of the \( z_i \)s in Eqs. (7) and (8) allows PixelSNE to pre-compute the information needed to find a single representative point of multiple data items before the main iterations of the gradient-descent updates (Fig. 1(c-1)).

To find a single representative point, each cell \( c \) of a quadtree should contain the following information:

1. The boundary \( B_d^{(c)} \) about its corresponding 2D region, i.e.,

\[
B_d^{(c)} = \left[ b_{d,\text{min}}^{(c)}, b_{d,\text{max}}^{(c)} \right] \quad \text{for} \; d = 1, 2
\]

(9)

2. The representative point \( z_c \) of \( z_i \)'s in a cell \( c \).

Firstly, the fixed range of \( z_i \) enables PixelSNE to compute all the cell boundaries, i.e., \( B_d^{(c)} \) of P-Quadtree before the main iteration. Since the range of \( z_i \) corresponds to a given screen resolution, i.e., \( \min_i z_{i,d} = 0 \) and \( \max_i z_{i,d} = r_d \), the boundary of a root node of P-Quadtree is determined as

\[
b_{d,\text{min}}^{(\text{root})} = 0 \quad \text{and} \quad b_{d,\text{max}}^{(\text{root})} = r_d + \epsilon, \; \forall d \in \{1, 2\},
\]

which is no longer dependent on iteration \( t \). This constant boundary of the root node makes those of all the child cells of P-Quadtree constant. Therefore, the boundary information can be reused as long as the low-dimensional data points are directly embedded onto the fixed screen space.

Next, for a representative point \( z_c \) of the points \( z_i \) in a cell \( c \), we simply approximate it as the center location of the corresponding cell region, i.e.,

\[
z_c = \left[ \frac{z_{c,1}}{z_{c,2}} \right], \quad \text{where} \quad z_{c,d} = \frac{b_{d,\text{min}}^{(c)} + b_{d,\text{max}}^{(c)}}{2}, \; \forall d \in \{1, 2\},
\]

(10)

which is also not dependent on iteration \( t \) because the boundary of each cell is already determined.

**Bounded tree depth based on precision parameter \( r \).** Another characteristic of our proposed P-Quadtree lies in that its depth is determined by the precision parameter \( r \). Whereas BH-SNE subdivides D-Quadtree until each leaf cell contains at most one data point (Fig. 1(T-1)), the size of the leaf node in P-Quadtree is bounded to the pixel size in order to avoid excessive precision in subsequent computations (Fig. 1(T-2)). Therefore, P-Quadtree continues splitting the cell \( c \) while satisfying the condition

\[
h_{d,\text{max}}^{(c)} - h_{d,\text{min}}^{(c)} > 1, \; \forall d \in \{1, 2\},
\]

(8)

which indicates that the length of a pixel-bounded cell is larger than the unit pixel size \( 1 \) for at least one of the two axes. As a result, the depth of P-Quadtree is bounded by

\[
\max_{d \in \{1, 2\}} \left\lfloor \log_2 r_d \right\rfloor.
\]

**Gradient-descent process using P-Quadtree.** Once the construction of P-Quadtree (Fig. 1(c-1)) is complete, the iterative gradient optimization in PixelSNE simply assigns \( z_i \)'s to the pre-computed cells in P-Quadtree and the following information is updated (Fig. 1(c-2)): (1) the set \( Z_c \) of \( z_i \)'s contained in \( c \), i.e.,

\[
Z_c = \left\{ z_i : z_i \in B_d^{(c)} \right\} \quad \text{for} \; d = 1, 2
\]

(11)

and (2) its cardinality \( N_c = |Z_c| \). PixelSNE assigns \( z_i \)'s to pre-computed cells until a cell contains at most one point or reaches the pixel-bounded leaf cells. Note that even though the information about the (pixel-bounded) leaf nodes is pre-computed, if an internal node contains only a single data point, then its child node is not used during the gradient-descent updates, as shown from dashed lines in Fig. 1.

Afterwards, PixelSNE traverses the tree via a depth-first search to find a single representative point \( z_c \) of the \( z_i \)'s contained in a particular cell \( c \). Since the minimum size of the cell in P-Quadtree is bounded to the pixel size, we may not find the cell satisfying the original criterion of BH-SNE, i.e., \( d_c / \|z_c - z\|_2 < \theta \), even after reaching a (pixel-bounded) leaf node \( c_{\text{leaf}} \). In this case, we simply stop the tree traversal and use the center location of this pixel-bounded cell to represent the points contained in this cell. This is described as:

\[
\begin{cases}
    z_c = \frac{d_c}{\|z_c - z_e\|_2} < \theta \\
    z_{c,\text{leaf}} \quad \text{otherwise},
\end{cases}
\]

(12)

With this information, PixelSNE approximates the term \(-d_{ij}^q Z(z_i - z_j)\) in Eq. (3) for those points contained in \( c \) as

\[-N_c d_{ij}^q Z(z_i - z_e)\]

substituting \( N_c \) number of multiple data points in cell \( c \), i.e., the \( z_i \)'s with \( z_c \).

**Computational complexity.** The depth of the Barnes-Hut tree acts as a critical factor in algorithm efficiency, since both the assignment of data point to cells and the approximation of \( F_{\text{rep}} \) (Eq. (3)) are performed based on depth-first search traversal (of which the computational complexity is linear to the tree depth). In the worst
Screen-Driven Scaling

Algorithm 1: Pixel-Aligned SNE

\[ \begin{align*}
\text{Input:} & \quad \text{Data set } X = \{x_i \in \mathbb{R}^d\}, i=1, \ldots, N, \\
& \quad \text{precision parameter } r_d, \forall d \in \{1, 2\}, \\
\text{Output:} & \quad 2D \text{ embedding } Z = \{z_i \in \mathbb{R}^2\}, i=1, \ldots, N
\end{align*} \]

1. Construct the high-dimensional joint probability matrix \( P \)
2. Initialize \( Z^{(0)} = \{z_i \in \mathbb{R}^2\}, i=1, \ldots, N \)
   where \( 0 \leq z_{i,d} \leq r_d, \forall d \in \{1, 2\} \)
3. Initialize \( \beta_d^{(0)} = \left(\frac{\pi}{r_d^2}\right)^2 \)
4. Construct P-Quadtree as in Fig. 1(c-1)
   e.g., compute \( B_d^{(0)} \) (Eq. (9)) and \( z_{i,d} \) (Eq. (10))
5. for \( i = 1 \) to \( T \) do
   Traverse the P-Quadtree to find the cell satisfying Eq. (12)
   Compute low dimensional probability matrix \( Q \) with Eq. (14)
   Perform gradient-descent updating \( Z^{(t-1)} \) to \( Z^{(t)} \)
   Perform screen-driven scaling on \( Z^{(t)} \) resulting in \( Z^{(t)} \) by Eq. (13)
   Update \( \beta_d^{(t)} \) according to Eq. (15)

end

\[ \begin{align*}
\text{end}
\end{align*} \]

where \( \beta_d^{(t)} = \sum_{d=1}^{2} \beta_d^{(t-1)} \frac{d_i^{(t-1)} - d_j^{(t-1)}}{d_i^{(t-1)} - d_j^{(t-1)}} \) and \( \beta_d \) is defined as
\[ \beta_d = \frac{1}{\prod_{s=1}^{2} d_i^{(t-1)}} \quad \forall d \in \{1, 2\}. \]
Table 1: Comparison of computing times among 2D embedding algorithms. The total computing times are shown in boldface, and the numbers in parentheses represent the standard deviations from three repetitions. The t-SNE results are excluded for large datasets owing to their long computing time.

<table>
<thead>
<tr>
<th></th>
<th>20News</th>
<th>FacExp</th>
<th>MNIST</th>
<th>NewsAgg</th>
<th>Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>Coord</td>
<td>Total</td>
<td>P</td>
<td>Coord</td>
</tr>
<tr>
<td>t-SNE</td>
<td>2.43m</td>
<td>(0.98s)</td>
<td>4.51m</td>
<td>6.48m</td>
<td>(0.34s)</td>
</tr>
<tr>
<td>BH-SNE</td>
<td>13.92s</td>
<td>(0.19s)</td>
<td>161.66s</td>
<td>175.52s</td>
<td>(3.05s)</td>
</tr>
<tr>
<td>PixelSNE-VP</td>
<td>14.68s</td>
<td>(1.46s)</td>
<td>70.71s</td>
<td>85.39s</td>
<td>(3.03s)</td>
</tr>
<tr>
<td>PixelSNE-RP</td>
<td>15.53s</td>
<td>(0.49s)</td>
<td>72.17s</td>
<td>87.70s</td>
<td>(1.93s)</td>
</tr>
</tbody>
</table>

5.1.1. Datasets

Our experiments used the following five real-world datasets: (1) MNIST digit images, (2) Facial expression images (FacExp), (3) 20 Newsgroups documents (20News), (4) News aggregator dataset (NewsAgg), and (5) Yelp reviews.

MNIST dataset (http://cs.nyu.edu/~roweis/data.html) contains a set of 70,000 grayscale handwritten digit images with 28 × 28 = 784 pixels, resulting in an input matrix \( \mathbf{X} \in \mathbb{R}^{784 \times 70000} \). Each image has its digit label as one of the 10 different digits.

FacExp dataset (https://goo.gl/W3Z8qZ/) contains a set of 28,709 grayscale facial images with seven labels for different facial expression categories. For pre-processing, we extracted the features from each image using a convolutional neural network model, which is trained for the task of classification with the seven categories. We used the last hidden layer output of 256 dimensions as the feature vector of each image, resulting in an input matrix \( \mathbf{X} \in \mathbb{R}^{256 \times 28709} \).

20News dataset (http://qwone.com/~jason/20Newsgroups/) is a collection of 18,846 newsgroup posts on 20 different newsgroup categories. We used seven higher-level newsgroup categories to label the data. For each document representation, we averaged a pre-trained word2vec embedding vector [MSC13] of those words in it, which results in an input matrix \( \mathbf{X} \in \mathbb{R}^{300 \times 18846} \).

NewsAgg dataset (https://www.kaggle.com/uciml/news-aggregator-dataset) contains a collection of 421,161 news headlines under four different categories. We represented each headline with the averaged words vector pre-processed as explained above. This resulted in an input matrix \( \mathbf{X} \in \mathbb{R}^{500 \times 421161} \).

Yelp review dataset (https://www.yelp.com/dataset_challenge/) contains around 2.7 million reviews. We removed the keywords that appeared in less than 20 documents and we also removed stop words. After forming a term-document matrix, we randomly selected one million reviews from it, resulting in an input matrix \( \mathbf{X} \in \mathbb{R}^{33706 \times 1,000,000} \). We extracted 10 topic labels through topic modeling with the method of nonnegative matrix factorization [LS99].

For all datasets, we reduced the dimensionality to 50 by applying PCA, which is a standard pre-processing step of t-SNE and BH-SNE.

5.1.2. Compared Methods

We compared our methods against the original t-SNE and BH-SNE. For both the methods, we used the publicly available code written by the original author (https://lvdmaaten.github.io/tsne/) We used the default parameter values for both methods, e.g., the perplexity value was 50, the number of iterations was 1,000, and the threshold \( \theta \) in Eq. (6) was 0.5.

For PixelSNE, we used two different versions depending on the algorithm for constructing P: (1) the vantage-point tree (PixelSNE-VP) used originally in BH-SNE and (2) the random projection tree-based approach (PixelSNE-RP), which we described in the previous section. For the latter, we exploited the publicly available code (https://github.com/lferry007/LargeVis/), with its default parameter settings but setting the number of threads to 1 in order to exclude the effect of parallelism. We set the number of iterations and the threshold \( \theta \) in Eq. (12) to the same as in BH-SNE. We set both the precision parameter \( r_1 \) and \( r_2 \) to 512 for the 20News and FacExp datasets. For MNIST, NewsAgg and Yelp datasets, they were 1024, 2048, and 8192, respectively.

5.2. Computing Time Analysis

In this section, we present the results of the four different experiments with regard to the computing time.

Method comparisons. Table 1 shows the comparison of computing times for various algorithms with the five different datasets. We report the computing time for the followings: (1) constructing the original pairwise probability matrix \( P(P) \), (2) optimizing low-dimensional coordinates (Coord), and (3) the total computing time (Total). Due to its significant computing time, the computation time results of t-SNE are excluded for the large datasets of the Yelp and NewsAgg datasets.

In all cases, PixelSNE-VP and PixelSNE-RP consistently outperformed t-SNE and BH-SNE by a large margin. For example, for...
the Yelp dataset, BH-SNE took almost 15 hours, while PixelSNE-VP and PixelSNE-RP took 12 hours and less than 4 hours, respectively. For optimizing the low-dimensional coordinates (Coord), we mainly applied our pixel-driven precision, PixelSNE-VP both showed a significant performance boost against BH-SNE. For instance, they computed this part almost three times faster than BH-SNE, it tended to run slowly as the size of the data got larger, e.g., NewsAgg and Yelp datasets, PixelSNE-RP ran increasingly faster than BH-SNE as well as t-SNE, which shows the promising scalability of PixelSNE.

Effects of the precision parameter. Fig. 3 shows the computing time depending on the precision parameter for 30,000 and 50,000 random samples from the MNIST dataset. As the data size got larger, our PixelSNE-RP ran increasingly faster than BH-SNE as well as t-SNE, which shows the promising scalability of PixelSNE.

5.3. Embedding Quality Analysis

In this section, we compare the overall quality of embedding between PixelSNE-RP and BH-SNE and investigate the effect of the precision parameter $r$.

Evaluation Measures To analyze the embedding quality we adopted the following two measures. (1) Neighborhood precision, which measures how many of the $k$ original nearest neighbors in a high-dimensional space, i.e., $N^k_D$, are captured in the $k$ nearest neighbors in the low-dimensional (2D) space, i.e., $N^k_d$. Similar to [KRM+17], the neighborhood precision was computed as

$$\frac{1}{kN} \sum_{i=1}^{N} |N^k_D(i) \cap N^k_d(i)|$$ (16)

where $N$ is the number of data. (2) $k$-NN classification accuracy, which measures the $k$-nearest neighbor classification accuracy based on the 2D embedding results along with their labels.

<table>
<thead>
<tr>
<th>$r$</th>
<th>$k$ in $k$ nearest neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>512</td>
<td>.9390</td>
</tr>
<tr>
<td></td>
<td>(.0012)</td>
</tr>
<tr>
<td>1024</td>
<td>.9387</td>
</tr>
<tr>
<td></td>
<td>(.0017)</td>
</tr>
<tr>
<td>2048</td>
<td>.9386</td>
</tr>
<tr>
<td></td>
<td>(.0020)</td>
</tr>
<tr>
<td>4096</td>
<td>.9380</td>
</tr>
<tr>
<td></td>
<td>(.0018)</td>
</tr>
</tbody>
</table>

Method comparisons. Fig. 4 shows the comparison results between PixelSNE-RP and BH-SNE depending on different $k$ values. For both evaluation measures, we observed that PixelSNE-RP achieved similar or better performance compared to BH-SNE. In Figs. 4(a), (c), and (d), even though our method compromises the precision for the sake of computing time, the performance gap between the two was similar, indicating that the outputs of our method have a comparable quality to that of BH-SNE. Also, notice that 4(b) and (e) show that PixelSNE-RP achieved better results than BH-SNE. We conjecture that the reason for this is that the random projection tree used in PixelSNE-RP finds more accurate nearest neighbors than the vantage-point tree used in BH-SNE [TLZM16].

Effects of the precision parameter. Tables 2 and 3 show the above
two measures for PixelSNE-RP with respect to different values of a precision parameter $r$. Unlike the k-NN classification accuracy, which stays roughly the same regardless of different values of $r$, the neighborhood precision consistently increased as $r$ increased; however, the gap was not very significant.

Table 4: Comparison of cost function values between BH-SNE and PixelSNE-RP. The numbers in parentheses represent the standard deviation from ten repetitions.

<table>
<thead>
<tr>
<th>Random Coordinates</th>
<th>BH-SNE</th>
<th>PixelSNE-RP (with $r$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinates</td>
<td>512</td>
<td>1024</td>
</tr>
<tr>
<td>Cost Value (Eq. 2)</td>
<td>91.580</td>
<td>(1.815)</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>(.000)</td>
<td>(.0071)</td>
</tr>
</tbody>
</table>

Cost function values comparison. Finally, Table 4 compares the cost function value (Eq. (2)) after convergence. Considering the baseline cost value of the random initialization, both BH-SNE and PixelSNE-RP achieved a similar level of the algorithm objective. Also, for all the values of $r$ tested, this value remained almost the same between BH-SNE and PixelSNE-RP, which indicates that the screen resolution-based precision of PixelSNE-RP had minimal to no impact in achieving the optimal cost value, which is consistent with the results found in Tables 2 and 3.

5.4. Exploratory Analysis

Finally, we present visualization examples of Yelp and NewsAgg datasets. Fig. 5 presents the visualization results of PixelSNE-RP and BH-SNE on Yelp datasets. Even though PixelSNE-RP ran outstandingly faster than BH-SNE, the visualization results are comparable to each other.

Fig. 6 shows the visualization results on the NewsAgg dataset from PixelSNE-RP. A data point, which corresponds to a news headline, is labeled with four different categories: Business, Science and Technology, Entertainment, and Health. Although all the headlines were categorized into only four subjects, Fig. 6 reveals the additional sub-category information by closely embedding news of similar topics. For example, the news belonging to Health category was roughly divided into three parts. The topical region “Infectious disease” on the left side reveals the headlines with keywords such as “Ebola,” “West Nile virus” or “Mers.” On the contrary, the region connected with “Non-infectious disease” had headlines with the keywords of “Breast cancer,” “Obesity,” or “Alzheimer.” Lastly, the topical area of “Drug,” “Tobacco” includes news about health-related life styles, e.g., “Mid-life Drinking Problem Doubles Later Memory Issues.” Note that the topical regions of “Infectious disease” and “Non-infectious disease” are closely located compared to the latter. This result demonstrates that PixelSNE preserved the high-dimensional data structure well when embedding them in a 2D space even with the pixel-based precision. Each topic was connected to the relevant area as shown in Fig. 6.
6. Conclusion

In this paper, we presented a novel idea for exploiting screen resolution-driven precision as the fundamental framework for significantly accelerating 2D embedding algorithms. Experiments on real-world datasets demonstrated that PixelSNE can easily scale up to large datasets while preserving the quality of embedding.

One limitation of our method is that users cannot obtain a more precise result in which they have an interest. To make future improvements in terms of user interactivity, we plan to enable users to dynamically control the precision of a local part of the embedding depending on their region of interest via a locally applicable precision parameter $r$.

Acknowledgments

This work was supported in part by the Korean National Research Foundation (NRF) grant funded by the Korean government (MSIP) (No. NRF-2016R1C1B2015924). Jaegul Choo is the corresponding author.

References


[MSC+13] Mikolov T., Sutskever I., Chen K., Corrado G. S., Dean J.: Distributed representations of words and phrases and their applications. Figure 6: 2D embedding of the NewsAgg dataset generated by PixelSNE-RP.


© 2018 The Author(s)

Computer Graphics Forum © 2018 The Eurographics Association and John Wiley & Sons Ltd.