

Using Dimensionality Reduction to Optimize t-SNE

Rikhav Shah

UC Berkeley, Berkeley, CA 94720

RIKHAV.SHAH@BERKELEY.EDU

Sandeep Silwal

Massachusetts Institute of Technology, Cambridge, MA 02139

SILWAL@MIT.EDU

Abstract

t-SNE is a popular tool for embedding multi-dimensional datasets into two or three dimensions. However, it has a large computational cost, especially when the input data has many dimensions. Many use t-SNE to embed the output of a neural network, which is generally of much lower dimension than the original data. This limits the use of t-SNE in unsupervised scenarios. We propose using *random* projections to embed high dimensional datasets into relatively few dimensions, and then using t-SNE to obtain a two dimensional embedding. We show that random projections preserve the desirable clustering achieved by t-SNE, while dramatically reducing the runtime of finding the embedding.

1. Introduction

Cluster analysis of high dimensional datasets is a core component of unsupervised learning and a common task in many academic settings and practical applications. A widely used practice to more easily visualize the cluster structure present in the data is to first reduce the dimension of data to a lower dimension, typically two. A popular tool to perform this dimensionality reduction is the t-distributed Stochastic Neighborhood Embedding (t-SNE) algorithm, introduced by van der Maaten and Hinton [11]. t-SNE has been empirically shown to preserve local neighborhood structure of points in many applications and the popularity of this method can be seen by the number of citations of the original paper (9000+) and implementations in popular packages such as scikit-learn.

The main drawback of t-SNE however, is its large computational cost, a big part of which can be written as $O(\# \text{ of distances computed} \cdot d)$ where d is the dimension of the input points. This drawback severely limits its potential use in unsupervised learning problems. To combat the large computational cost of t-SNE, many practical developments have been made using techniques such as parallel computation, GPU, and multicore implementations with a large chunk of work on reducing the number of pairwise distances that are computed between the input points [1, 7, 10]. We take a different approach by focusing instead on reducing the value of d in the runtime cost.

In this work, we show that the distance computation can be optimized by first projecting the input data into a much smaller random subspace before running t-SNE. Our motivation for this approach comes from the field of metric embeddings where it is known that random projections preserve many geometric structures of the input data. For more information, see Section 1.3. More formally, our contributions are the following:

- We empirically show that projecting to a very small dimension (even as low as 5% of the original dimension in some cases) and then performing t-SNE preserves local cluster information just as well as t-SNE with no prior dimensionality reduction step.

- We empirically show that performing dimensionality reduction first and then running t-SNE is significantly faster than just running t-SNE.

We note that our last result is implementation agnostic, since every implementation must compute distances among some of the input points. To show the benefits of our approach and simplify our experiments, we use two implementations of t-SNE: a widely available, but slower scikit-Learn implementation, and a highly optimized Python implementation called ‘openTSNE’ [8]. For more details about our experimental setup, see Section 2.1.

1.1. Measuring Quality of an Embedding

We devise an *accuracy score* to measure the quality of the cluster structure of an embedding. First we note that ideal clusters are ones in which every element has the same label as others in the same cluster. If one has labels for only a few datapoints, this kind of clustering would allow much of the dataset to be correctly labeled. Thus ideally, such clustering would be created without using labels (note that neither random projections nor t-SNE use labels to determine embeddings). A datapoint is said to be “correctly clustered” in the low-dimension embedding if it’s label matches that of its nearest neighbor. The *accuracy score* of an embedding is given by the fraction of datapoints that are correctly clustered. Such a score could be refined by instead considering the modal label of the k -nearest-neighbors. The accuracy score rewards the case that clusters in high dimension stay together in the lower dimension and penalizes the case of different clusters in the higher dimension merging together in the lower dimension since then we can expect the nearest neighbors of many points to have a differing label. In Section 2.2, we compare the accuracy scores of performing dimensionality reduction and then t-SNE versus performing only t-SNE on various labeled datasets.

1.2. Overview of t-SNE

Given a set of N points $x_1, \dots, x_N \in \mathbb{R}^d$, t-SNE first computes the similarity score p_{ij} between x_i and x_j defined as $p_{ij} = (p_{i|j} + p_{j|i})/(2N)$ where for a fixed i , $p_{j|i} \propto \exp(-\|x_i - x_j\|^2/\sigma_i^2)$ for some parameter σ_i . Intuitively, the value p_{ij} measures the ‘similarity’ between points x_i and x_j . t-SNE then aims to learn the lower dimensional points $y_1, \dots, y_N \in \mathbb{R}^2$ such that if $q_{ij} \propto (1 + \|y_i - y_j\|_2^2)^{-1}$, then q_{ij} minimizes the Kullback–Leibler divergence of the distribution $\{q_{ij}\}$ from the distribution $\{p_{ij}\}$. For a more detailed explanation of the t-SNE algorithm, see [11].

1.3. Motivation for Using Random Projections

Like t-SNE, using a random projection is also a dimensionality reduction tool which has been shown to preserve many geometrical properties of the input data, such as pairwise distances, nearest neighbors, and cluster structures [3, 4, 6]. One of the key results of this field is the well known Johnson–Lindenstrauss (JL) Lemma which roughly states that given any set of N points, a random projection of these points into dimension $O(\log N/\epsilon^2)$ preserves all pairwise distances upto multiplicative error $(1 \pm \epsilon)$ [3]. However, it is not possible to use a random projection to project down to a very small subspace, such as two dimensions used in most applications, while still maintaining the desired geometric properties. This is in contrast to t-SNE which has been empirically shown to preserve many local properties of the input data even in very low dimensions (such as two). However, we can leverage advantages of both of these approaches through the counter intuitive idea of using dimensionality reduction *before* dimensionality reduction. More specifically, we can hope

that by first randomly projecting the input data into a smaller dimension, we can preserve enough local information for t-SNE to use to be able to meaningfully project the data into two dimensions and still maintain the inherent cluster structure.

Indeed, if we consider the similarity scores p_{ij} computed by t-SNE, we see that if a random projection preserves pairwise distances, then the values of the similarity scores remain the same. Furthermore, if we require the weaker condition that the random projection preserves nearest neighbor relations, then we see that the values p_{ij} maintain their order. More precisely, if $\|x_i - x_j\| < \|x_i - x_k\|$ holds, the similarity scores between points x_i and x_j will be large and thus, t-SNE will try to place these points closer together. Using a random projection from \mathbb{R}^d to \mathbb{R}^k , we immediately get a reduction of the runtime from $O(\# \text{ of distances computed} \cdot d)$ to $O(\# \text{ of distances computed} \cdot k)$. We show the success of our method in Section 2.

1.4. Related Works

There have been many works about improving t-SNE through better implementations, such as GPU acceleration or parallel implementations [1, 7, 9]. There has also been work on reducing the number of distances that need to be computed using tree based methods, see [10] for more info. To our knowledge, our work is the first to suggest using random projections in conjunction with t-SNE. We have verified this claim by going through the papers that cite the original t-SNE paper of [11] and searching Google Scholar for papers with the phrase ‘random projection’.

2. Experimental Results

2.1. Experimental Setup and Justification

We construct embeddings by first applying a random projection into d' dimensions, then using t-SNE to arrive at a two dimensional embedding. The random projections are performed by using an appropriately sized random matrix with i.i.d. Gaussian entries $\mathcal{N}(0, 1/d)$ where d is the original dimension of the input data. We repeat this process for several values of d' exponentially spaced and ranging from 7 to d .

We use the following four datasets in our experiments: MNIST, Kuzushiji-MNIST (KMnist) (data set of Japanese cursive characters in the same format as MNIST) [2], Fashion-MNIST (data set of fashion images in the same format as MNIST) [12], and the Small Norb data set (data set of images of toys) [5]. All of our experiments were done in Python 3.7 using one core of a MacBook Pro using a 2.7 GHz Intel Core i5 processor. We only report the time to perform t-SNE since the t-SNE runtime is many orders of magnitude larger than all the other steps. Our code can be accessed from the following GitHub repository: <https://github.com/ssilwa/optml>.

2.2. Results

We first compare the time taken to perform t-SNE and the accuracy scores achieved if a dimensionality reduction step is used before using t-SNE versus the case where no dimensionality reduction is used. We do so by plotting the ratio of the runtime and accuracy scores. For a given dimension, a higher accuracy score ratio is better and signifies that projecting down to that dimension using a random projection does not deteriorate the performance of t-SNE. Likewise, a lower time ratio is better. In Figure 1 we plot the ratio of the accuracy scores and the time taken when we use the openTSNE implementation. The x axis is the dimension of the random projection (ranging all the

way up to the actual dimension of the input data). In all four datasets, we see that as the dimension increases, both the accuracy ratio and the ratio of the time taken approach 1. However, we also observe that the accuracy score ratios approach 1 much faster, indicating a ‘sweet spot’ where the dimension is high enough for the random projection to preserve geometric structure, while still low enough for t-SNE to run relatively quickly.

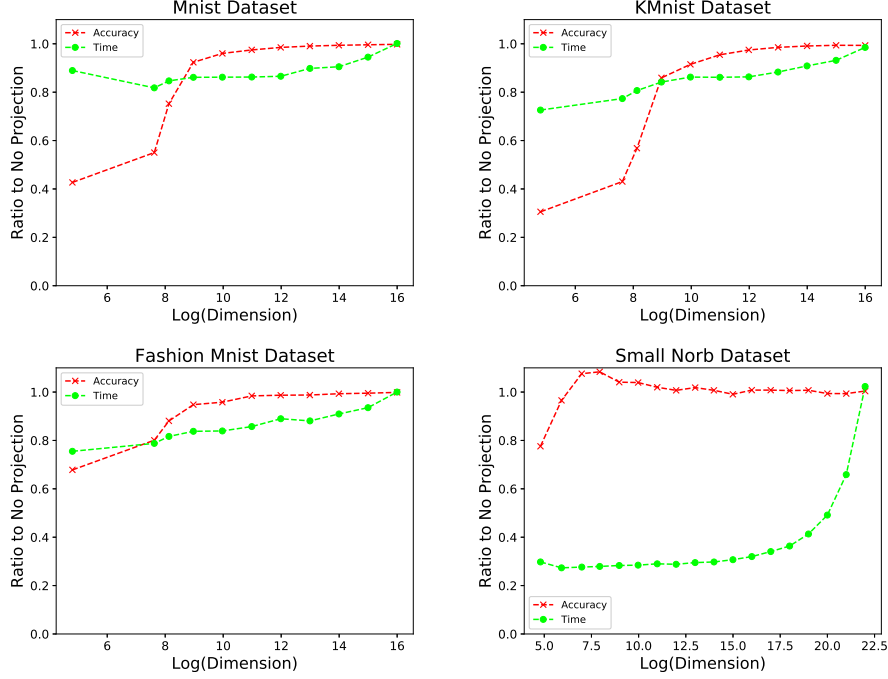


Figure 1: Green: Ratio of time taken to run t-SNE after dimensionality reduction to time taken to run t-SNE without dimensionality reduction. Red: Ratio of accuracy score after dimensionality reduction to accuracy score without dimensionality reduction. The t-SNE implementation used was openTSNE. x axis shows the dimension after dimensionality reduction. The base of the logarithm is 1.5.

Likewise, we show the results of the same experiments using the scikit-learn implementation. Since the scikit-learn implementation is significantly slower than the openTSNE implementation, we subsample the d values used and do not test the Small Norb dataset. However, even in this different implementation, we again observe the same trends as in the openTSNE implementation.

In practice, SVD methods such as principal component analysis (PCA) are more widely used than random projections for the general task of dimensionality reduction. We show that for our four datasets, PCA indeed outperforms random projections. That is, we empirically observe that using PCA, rather than a random projection before t-SNE, allows us to project to a much smaller dimension while still retaining a high accuracy score ratio, (compared to the case where no dimensionality reduction is used). This is shown in Figure 3 where regardless of the dataset, a projection to a dimension of $d = 25$ is sufficient to get an accuracy score ratio close to 1. An advantage of random projections however, is that it is *data oblivious* (the dimensionality reduction does not depend on the data) and it has provable guarantees in many cases, such as the JL lemma.

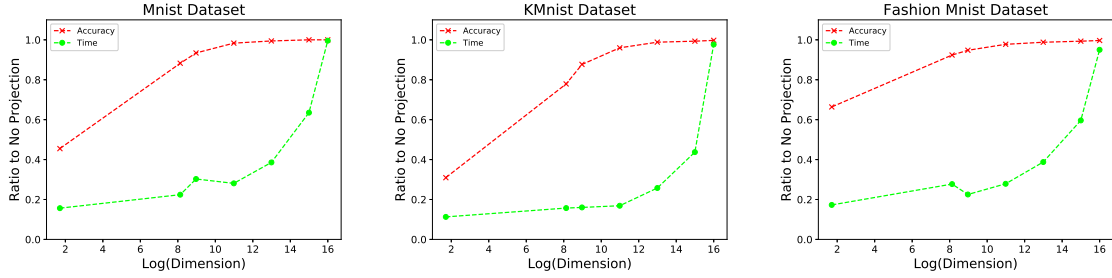


Figure 2: Green: Ratio of time taken to run t-SNE after dimensionality reduction to time taken to run t-SNE without dimensionality reduction. Red: Ratio of accuracy score after dimensionality reduction to accuracy score without dimensionality reduction. The t-SNE implementation used was the scipy implementation. x axis shows the dimension after dimensionality reduction. The base of the logarithm is 1.5. Since the scipy implementation is very slow, we used less datapoints for the datasets shown and did not test on the Small Norb dataset.

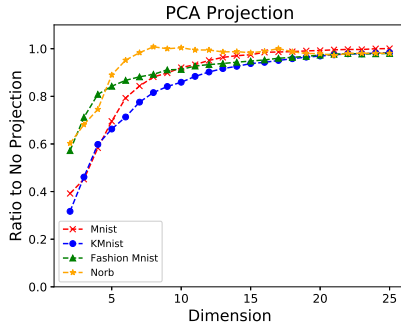


Figure 3: Ratio of the accuracy score using PCA for dimensionality reduction instead of random projections and then performing t-SNE. We see that across all datasets, the accuracy scores become comparable to the scores when using t-SNE with no prior dimensionality reduction even for very small dimensions. This suggests that in practice, PCA might be a better choice than random projections to perform dimensionality reduction before t-SNE.

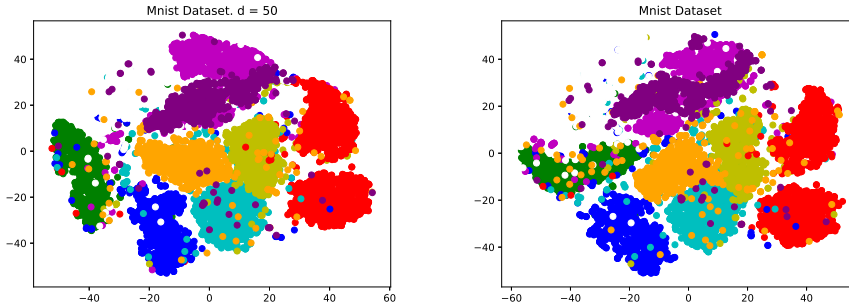


Figure 4: Plotting the output of t-SNE. Left: a dimensionality reduction to $d = 50$ is performed before t-SNE on the MNIST dataset. Right: no dimensionality reduction is performed. Colors represent the label of the points. Observe that the same cluster structures appear in both plots.

References

- [1] David M Chan, Roshan Rao, Forrest Huang, and John F Canny. Gpu accelerated t-distributed stochastic neighbor embedding. *Journal of Parallel and Distributed Computing*, 131:1–13, 2019.
- [2] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018.
- [3] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, January 2003. ISSN 1042-9832. doi: 10.1002/rsa.10073. URL <http://dx.doi.org/10.1002/rsa.10073>.
- [4] Piotr Indyk and Assaf Naor. Nearest-neighbor-preserving embeddings. *ACM Trans. Algorithms*, 3(3), August 2007. ISSN 1549-6325. doi: 10.1145/1273340.1273347. URL <http://doi.acm.org/10.1145/1273340.1273347>.
- [5] Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR’04*, pages 97–104, Washington, DC, USA, 2004. IEEE Computer Society. URL <http://dl.acm.org/citation.cfm?id=1896300.1896315>.
- [6] Konstantin Makarychev, Yury Makarychev, and Ilya Razenshteyn. Performance of johnson-lindenstrauss transform for k-means and k-medians clustering. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 1027–1038, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6705-9. doi: 10.1145/3313276.3316350. URL <http://doi.acm.org/10.1145/3313276.3316350>.
- [7] N. Pezzotti, J. Thijssen, A. Mordvintsev, T. Höllt, B. Van Lew, B. P. F. Lelieveldt, E. Eise-mann, and A. Vilanova. Gpgpu linear complexity t-sne optimization. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2019. doi: 10.1109/TVCG.2019.2934307.
- [8] Pavlin G. Poličar, Martin Stražar, and Blaž Zupan. opentsne: a modular python library for t-sne dimensionality reduction and embedding. *bioRxiv*, 2019. doi: 10.1101/731877. URL <https://www.biorxiv.org/content/early/2019/08/13/731877>.
- [9] Dmitry Ulyanov. Multicore-tsne. <https://github.com/DmitryUlyanov/Multicore-TSNE>, 2016.
- [10] Laurens van der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research*, 15:3221–3245, 2014. URL <http://jmlr.org/papers/v15/vandermaaten14a.html>.
- [11] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL <http://www.jmlr.org/papers/v9/vandermaaten08a.html>.
- [12] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.