CS267 Project Proposal: Computationally Efficient t-SNE Using the Fast Multipole Method

Roshan Rao, Forrest Huang, and David Chan

1 Introduction

Visualizing high dimensional data is a fundamental problem in machine learning. Because datasets often lie in high dimensional spaces, it can be difficult for users to gain conceptual understandings of the local and global structures of the data. In this project, we focus on t-SNE (T-Distributed Stochastic Neighbor Embedding) [6], which has recently gained traction in machine learning and other science communities [3]. t-SNE focuses on preserving the local structure of the data by attempting to accurately model pairwise distances from the high dimensional space in the lower-dimensional projected space.

A major barrier preventing t-SNE from widespread use is the computational cost associated with it. While PCA can quickly compute a lower dimensional embedding of the entire MNIST dataset (in under a minute), t-SNE implementations require a minimum of 10 minutes to perform the same computation. To help solve these computational inefficiencies, researchers perform PCA on their data prior to the t-SNE computation to reduce the complexity of distance calculations. While this may improve performance, it is likely that some local structure will be lost, leading to inaccurate clustering.

This project proposes a thorough investigation of the parallelization of t-SNE, and the construction of a fast approximation algorithm to extend the use of t-SNE to large scale, high dimensional datasets in a faithful way.

2 Related Work

2.1 t-SNE

t-Distributed Stochastic Neighbor Embedding (or t-SNE) [6] is a technique for dimensionality reduction that is particularly well suited to visualizing high-dimensional data, as it focuses on modeling local structure in the data using optimization techniques. To compute this embedding, we begin by converting the distances between data-points into conditional probabilities $p_{j|i}$ representing the probability that point x_i would pick point x_j to be its neighbor (if neighbors were picked in proportion to their probability density under a Gaussian distribution centered at x_i. Mathematically, this is given by

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

where σ_i is the variance of the centered Gaussian. Because these conditional probabilities represent pairwise structure in the data, we want to create a similar distribution in the lower dimensional space. If we try to replicate this distribution for the embedded points y_i and y_j in the lower dimensional space, we can suffer from the crowding problem: If we want to faithfully model the distances between nearby data points, most of the points that are a moderate distance will be too far away. In Symmetric SNE (the version where we use a Gaussian), the spring connecting the data point i to each of the distant map points will have a small attractive force. Even though these forces are small, the large number of such forces pushes distant data points into the center of the map, preventing gaps between natural clusters from forming. To solve this problem, t-SNE defines the distribution in the lower dimensional space using a student t distribution with one degree of freedom, giving it relatively heavy tails and compensating for the mismatched dimensionality (For more information, see [6]). Thus, we have:

$$q_{j|i} = \frac{(1 + ||y_i - y_j||^2)^-1}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

We then solve the natural optimization minimizing the KL divergence between P and Q over the points i, j in the projected space:

$$\underset{i,j \in Y}{\arg\min} \quad C = KL(P||Q) = \sum_i \sum_j p_{i|j} \frac{p_{i|j}}{q_{i|j}}$$

To solve this optimization problem, we randomly initialize the points in the low dimensional manifold, and iterate the gradient until convergence:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{i|j} - q_{i|j})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$$

Thus, the computation breaks down into an N-body particle simulation with attractive and repulsive forces [8]. Each step, thus, takes $O(n^2)$ time, making the computation rather slow.

2.2 Relation to N-Body problems

[8] show the relation between t-SNE and N-Body problems, then use this to apply the Barnes-Hut approximation to obtain an approximate $O(N \log N)$ algorithm, which allows t-SNE to scale to datasets of moderate size. There

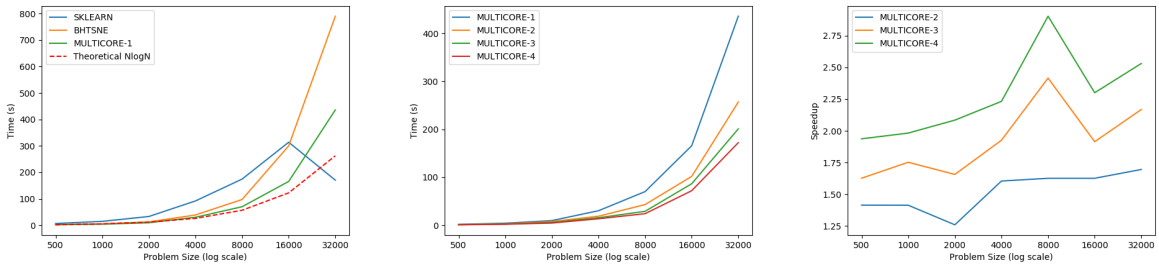

Figure 1: Initial benchmarking of Barnes-Hut t-SNE. Benchmarking performed on random data to simulate a worst-case scenario. Left: comparison of different single-threaded implementations and a theoretical NlogN algorithm. Note that the behavior of scikit-learn's t-SNE implementation is extremely odd as it actually decreases in overall time for 32k points. It may be performing some cutoff if progress is not detected, although we attempted to control this using the available arguments. Further analysis needs to be done to determine the actual cause of this behavior. Middle: Running time of multithreaded implementation provided by [7]. Right: Speedup of multithreaded implementation provided by [7].

are several other approximations of N-Body particle simulations that could result in an approximate $O(N)$ algorithm. One, which we implemented in class, employs a cutoff distance beyond which forces are assumed to be zero. Another is the Fast Multiple Method (FMM), although [8] and [9] claim that t-SNE would be hard to compute with this method because the forces are governed by a Student T-Distribution.

3 Proposed Method

In this project, we propose performing a number of steps to improve the performance of the t-SNE algorithm. These are enumerated below.

1. Benchmark existing t-SNE implementations, and determine the performance of such algorithms on large datasets. Figure 1 contains preliminary results of this benchmarking.

2. [8] showed that t-SNE could be accelerated using tree-based algorithms such as Barnes-Hut. We propose implementing these algorithms on the GPU, and comparing the performance to current state of the art non-parallel implementations (such as the implementation provided by Scikit-Learn), and parallel implementations such as those provided by [7] and [2]. Neither implementation provides a full parallelization with all parameters supported.

3. We propose implementing a local-forces approximation of the t-SNE N-Body problem with CUDA, and understand the performance and modeling trade-offs that would be made by using only a local-forces approximation to the original t-SNE.

4. While the Student T-Distribution may be difficult to model with FMMs, [4] suggests that by increasing the degrees of freedom of this distribution with a clever optimization trick, you can both obtain better visualizations than in the original t-SNE and avoid the crowding problem. Interestingly, increasing the degrees of freedom also means that we can approximate the T distribution with a suitable Gaussian, opening up the algorithm to FMM techniques. There are also black-box FMM methods that assume nothing about the force computation kernel. We propose exploring such approximations to improve the performance of the t-SNE algorithm.

The tasks above are ranked in order of ease of completion. Implementing a full-featured version of the Barnes-Hut algorithm on CUDA as described in [8] is the primary focus of the project and forms an extremely useful contribution to the current body of tools in the community.

4 Conclusion

We propose to implement a GPU-accelerated version of Barnes-hut t-SNE. t-SNE is a high impact algorithm and is widely used in many scientific communities. A GPU implementation would allow researchers who have access to a GPU but not a computing cluster to quickly perform t-SNE analysis on large datasets. In addition, the t-SNE algorithm is widely used by our lab (overseen by John Canny), and this would provide a direct speedup of our research. We also propose two possible extensions to improve the asymptotic performance of the algorithm from $O(N \log N)$ to $O(N)$, which would be a major result and open the door to t-SNE research on large datasets such as those used in vision (ImageNet [1], MSCOCO [5]), neuroscience [3], and security.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[2] G. Dimitriadis. T-sne-bhcuda. `https://github.com/georgedimitriadis/t_sne_bhcuda`, 2016.

[3] G. Dimitriadis, J. Neto, and A. Kampff. T-sne visualization of large-scale neural recordings. *bioRxiv*, 2016.

[4] J. Kitazono, N. Grozavu, N. Rogovschi, T. Omori, and S. Ozawa. t-distributed stochastic neighbor embedding with inhomogeneous degrees of freedom. In *International Conference on Neural Information Processing*, pages 119–128. Springer, 2016.

[5] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollr, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, Zrich, 2014. Oral.

[6] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[7] D. Ulyanov. Multicore-tsne. `https://github.com/DmitryUlyanov/Multicore-TSNE`, 2016.

[8] L. Van Der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of machine learning research*, 15(1):3221–3245, 2014.

[9] M. Vladymyrov and M. Carreira-Perpinan. Linear-time training of nonlinear low-dimensional embeddings. In *Artificial Intelligence and Statistics*, pages 968–977, 2014.