t-SNE-CUDA: GPU Accelerated t-SNE and its Applications to Modern Data

David Chan*, Roshan Rao*, Forrest Huang#* and John F. Canny
University of California, Berkeley

Presenter
* Equal contribution

Talk Outline

- The Problem
- t-SNE Overview
- Contributions
- Results
- Applications and Future Work

The Problem

The Problem

Source: ImageNet

Source: Eekim on Wikipedia

ImageNet
(> 1M Images)

GLoVE Vectors
(> 2.2M Words, 300-dimensions)

Modern machine learning datasets are massive, and high-dimensional

The Problem

Source: ImageNet

Source: Eekim on Wikipedia

ImageNet
(> 1M Images)

GLoVE Vectors
(> 2.2M Words, 300-dimensions)

How do we obtain a structural understanding of these datasets?

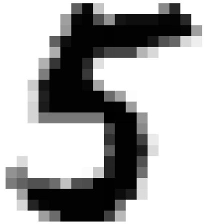
t-SNE Overview

(T-Distributed Stochastic Neighbor Embedding)

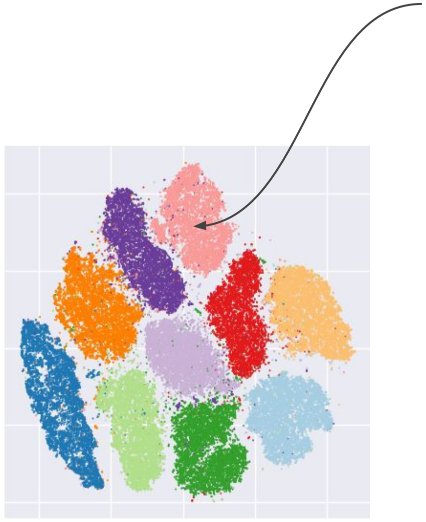
t-SNE Overview

- Dimension Reduction: We're given a set of elements $X_1 \ldots X_N$ in a high dimensional space, and we want to visualize them in a lower dimension as $Y_1 \ldots Y_N$

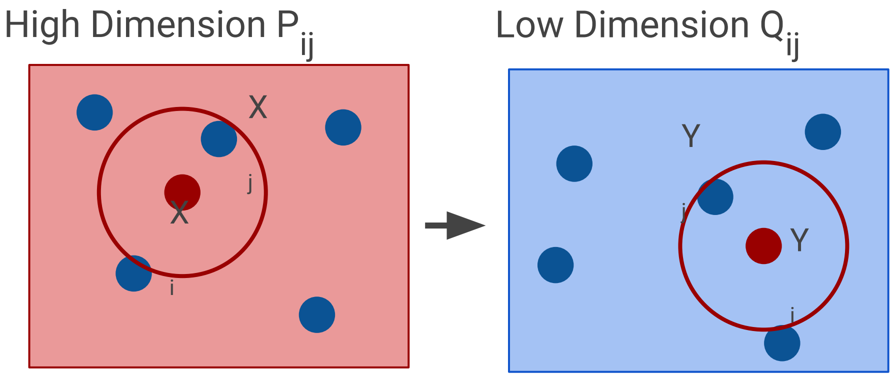
- t-SNE attempts to preserve between-point distances (as opposed to PCA, which preserves variance), leading to informative local structure



Source: MNIST



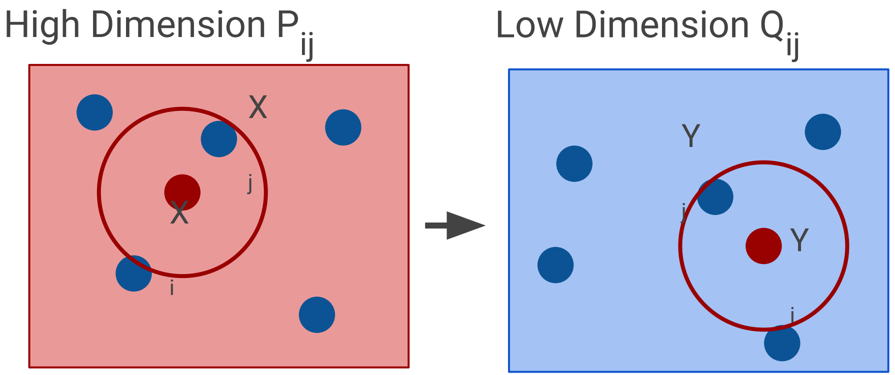
t-SNE Overview



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

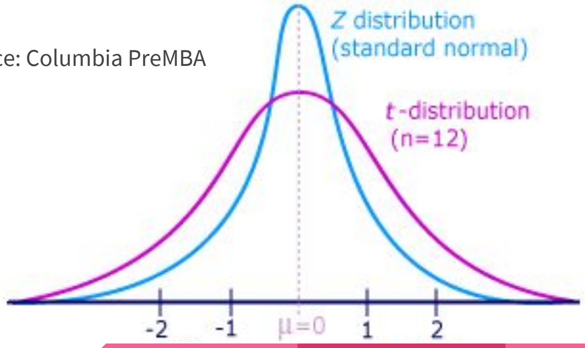
- Measure the similarity between points as a probability distribution
- Model the probability that X_i selects X_j as a neighbor from the larger set of points by using a gaussian centered at the point in high-dimensional space.

t-SNE Overview



- We map this distribution in the lower dimensional space on Y.
- t-Distribution in the lower dimensional space to avoid overcrowding problem with a heavier tail:

Source: Columbia PreMBA



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

t-SNE Overview

- Higher dimensional space:

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

t-SNE Overview

- Higher dimensional space:

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

t-SNE Overview

- Higher dimensional space:

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

- Lower dimensional space:

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

t-SNE Overview

- High dimensional:

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

- Low dimensional:

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

- Minimize KL Divergence:

$$y* = \arg\min_y KL(P||Q) = \arg\min_y \sum_{i,j,i \neq j} p_{ij} \log \frac{q_{ij}}{p_{ij}}$$

Implementation - Algorithm

- KL Divergence:

$$y* = \arg\min_y KL(P||Q) = \arg\min_y \sum_{i,j,i\neq j} p_{ij} \log \frac{q_{ij}}{p_{ij}}$$

- Minimize:

$$C = KL(P||Q)$$

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij}) q_{ij} (\mathbf{y}_i - \mathbf{y}_j)\, Z$$

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

Implementation - Algorithm

- KL Divergence:

$$y* = \arg\min_y KL(P||Q) = \arg\min_y \sum_{i,j,i \neq j} p_{ij} \log \frac{q_{ij}}{p_{ij}}$$

- Rewrite the gradient:

$$\frac{\partial C}{\partial y_i} = F_{attr} + F_{rep}$$

Implementation - Algorithm

- KL Divergence:

$$y* = \arg\min_{y} KL(P||Q) = \arg\min_{y} \sum_{i,j,i\neq j} p_{ij} \log \frac{q_{ij}}{p_{ij}}$$

- Rewrite the gradient:

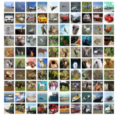
$$\frac{\partial C}{\partial y_i} = F_{attr} + F_{rep}$$

- Turn it into an N-body physics simulation, we can use Barnes-Hut method for efficient computation.

Issue

Source: ImageNet

ImageNet (1M Images)



Source: CIFAR-10

CIFAR-10
(60000 Images, 32 x 32 x 3)

Existing t-SNE implementations are slow: ImageNet would take more than a week to compute

Contributions/Solution

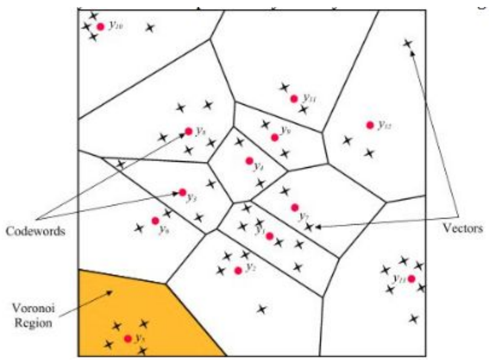
Solution - GPU Implementation

- GPU computation of t-SNE breaks down into several steps:
  - Computation of P_{ij}
  - Product between P_{ij} and Q_{ij}
  - Attractive forces
  - Building tree for Barnes-Hut Method for repulsive forces
  - Traversing tree for repulsive forces
  - Applying forces to the points in lower dimensional space
- We implement each of these kernels on the GPU
- Additionally, we add several other optimizations to further speed up the result
- We also provide python bindings for ease of use

Solution - Pij Construction

$$F_{attr} = \sum_{j \in [1,\ldots,N], j \neq i} p_{ij} q_{ij} Z(\mathbf{y}_i - \mathbf{y}_j)$$

- Usually, p_{ij} estimated by only computing k-nearest neighbours (k-NN) of each point (typically 32 points)
- **Improve with *approximate* k-NN using Product Quantization**
- **Use FAISS library**
    - Entirely on GPU
    - Can handle millions of points



Solution - Attractive Forces

- cuSPARSE sparse matrix computation:

$$F_{attr} = 4N((P_{ij} \odot Q_{ij})O \odot Y - (P_{ij} \odot Q_{ij})Y)$$

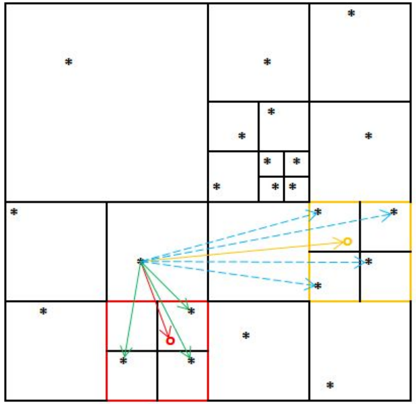
- O(kN) time with 5 operations; k = constant number of neighbours
  - 2 Hadamard Products,
  - 2 Matrix-Matrix Multiplications
  - 1 Matrix-Matrix Subtraction
- P_{ij} is a sparse matrix with O(kN) elements, so result can be computed efficiently with optimized cuSPARSE calls
- P_{ij} is fixed over duration of optimization, only Q_{ij}, Y change

Solution - Repulsive Forces

$$F_{rep} = -\sum_{j \in [1,\ldots,N], j \neq i} q_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)$$

Requires repeated computation of q_{ij} at each iteration

- Barnes-Hut quad-tree construction and traversal done in parallel on GPU
  - *O(NlogN)* time



Results

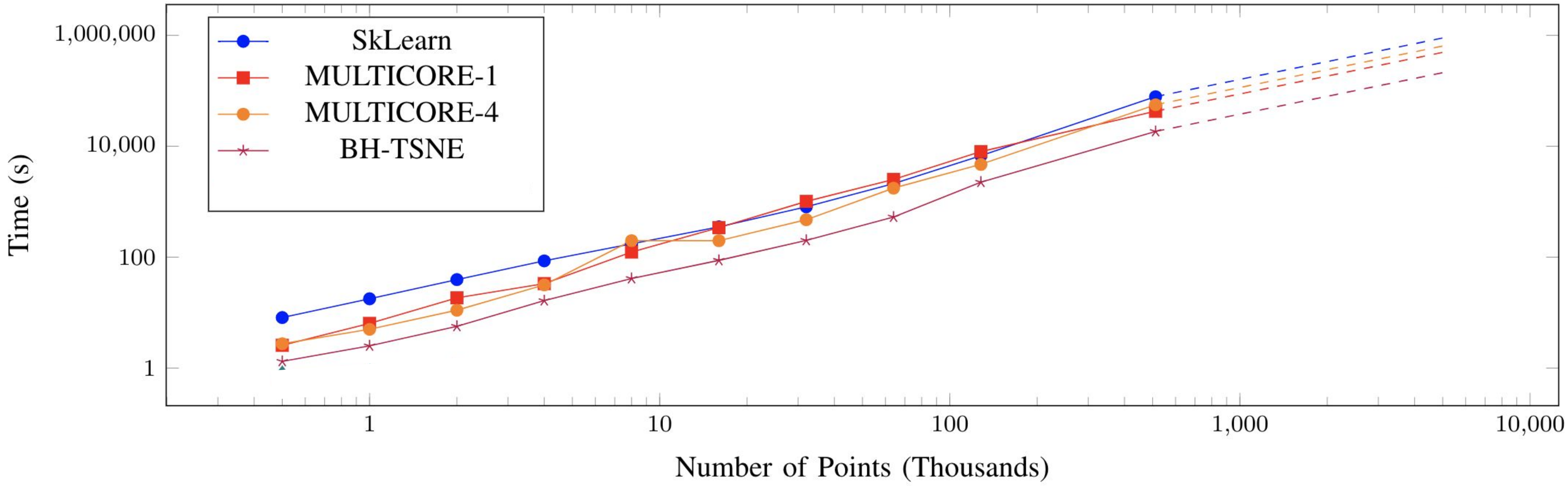
Results - Datasets

Simulated Data with 50-dimensional points sampled from 4 isotropic high-dimensional Gaussian distributions

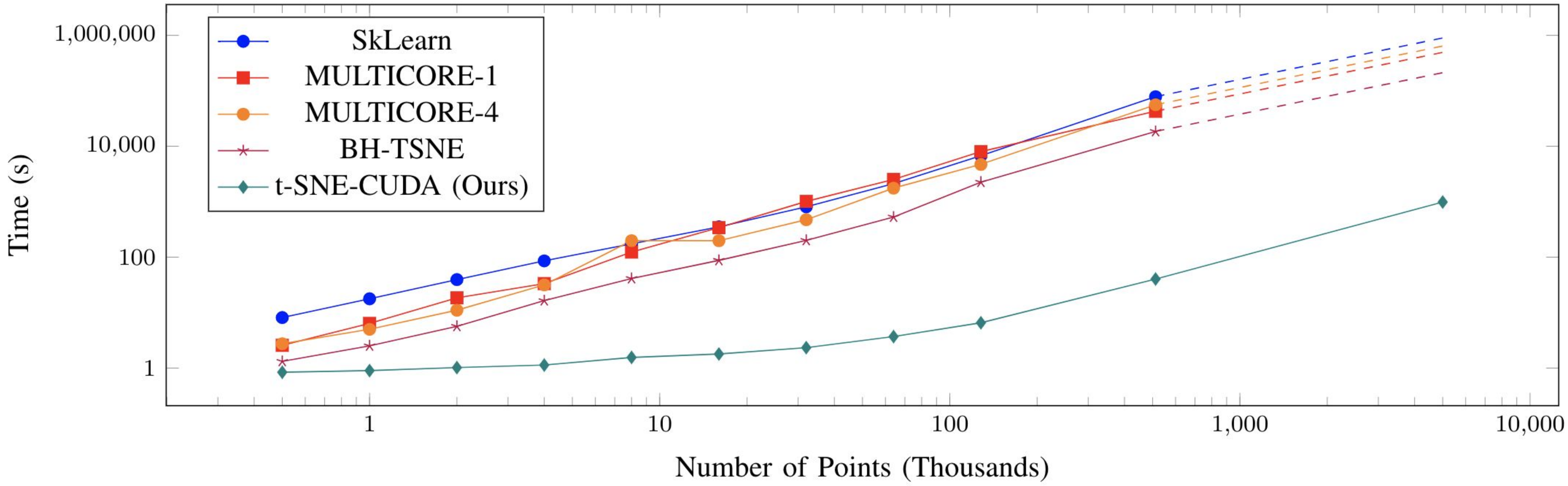
MNIST consisting of 70,000 28x28 images of handwritten digits constituting a *784* dimensional image space

CIFAR-10/100 datasets both consist of a 60,000 full-color images of 10 classes in 32x32x3 resolution, giving a *3072* dimensional data space

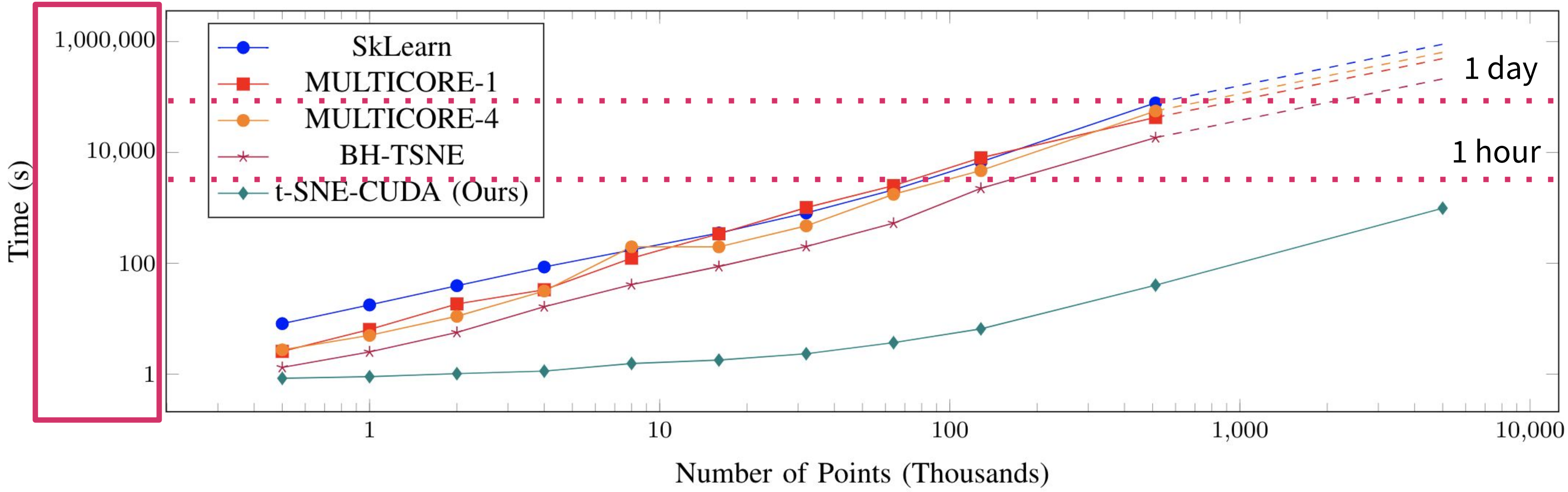
Results - Simulated Data



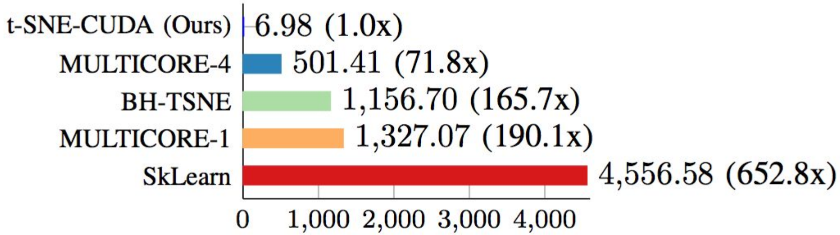
Results - Simulated Data



Results - Simulated Data



Results - MNIST

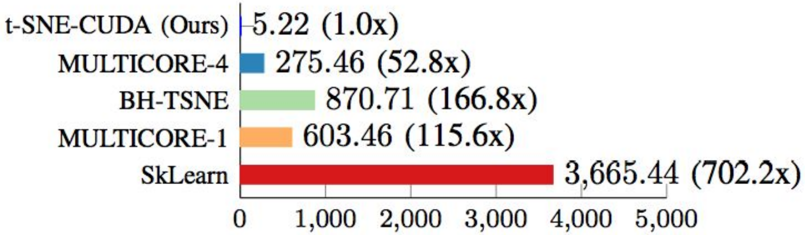


Running Time (seconds)

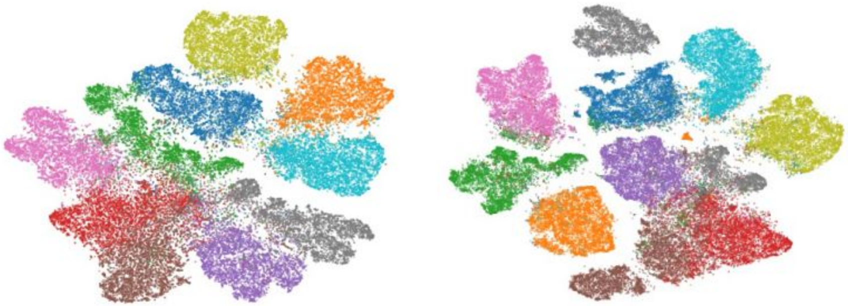
Visualization Results

72x - 650x Speedup

Results - CIFAR



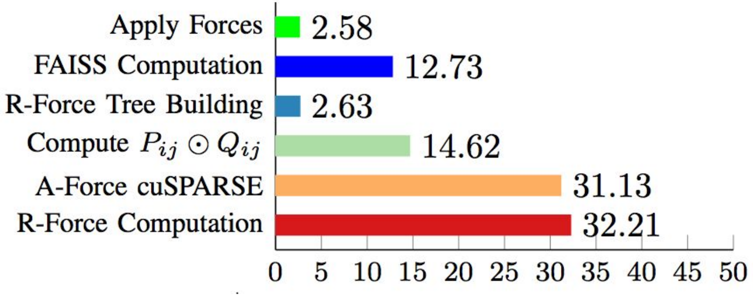
Running Time (seconds)



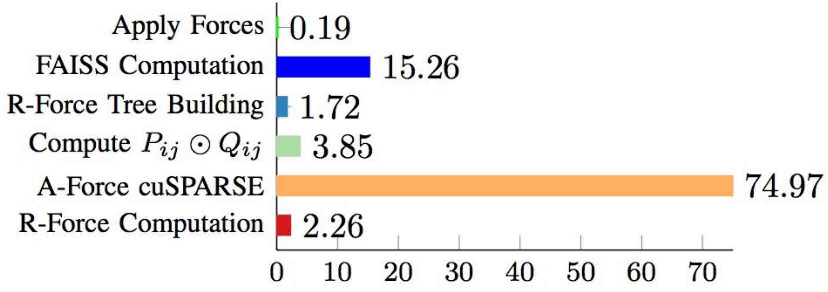
Visualization Results

52x - 700x Speedup

Results - Kernel Performance



500,000 points



5,000,000 points

cuSPARSE call dominate for large number of points

Results - Kernel Performance

- Floating-point focused kernel achieves 68.23% peak throughput, close to GPU roof-line
- Many computations are not floating-point arithmetics that GPUs are optimized for:
  - Many Kernels are memory/offset computations and transforms
  - Sparse Multiplication spends large amount of time to compute offsets for the different indices
- Kernels bounded by memory operations

Applications and Future Work

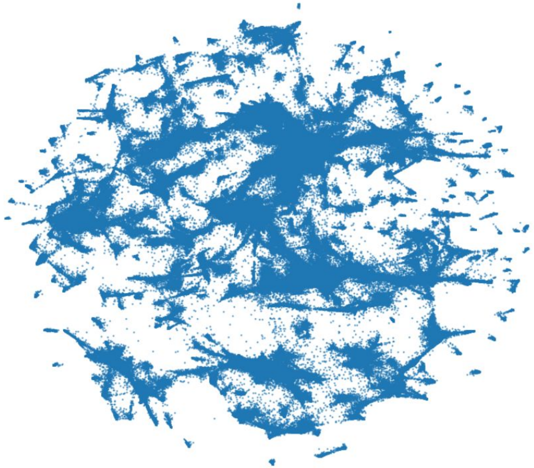
Applications - CIFAR-10 - Raw Pixels

- Raw pixel space doesn't exhibit structure
- Demonstrates the difficulty of classifying object classes in CIFAR-10 compared to MNIST:

Applications - CIFAR-10 - LeNet Code

- Raw pixel space doesn't exhibit structure
- Easier to classify object classes in CIFAR-10 in the LeNet code space, which demonstrates the effectiveness of Convolutional Neural Networks

Applications - ImageNet Codes

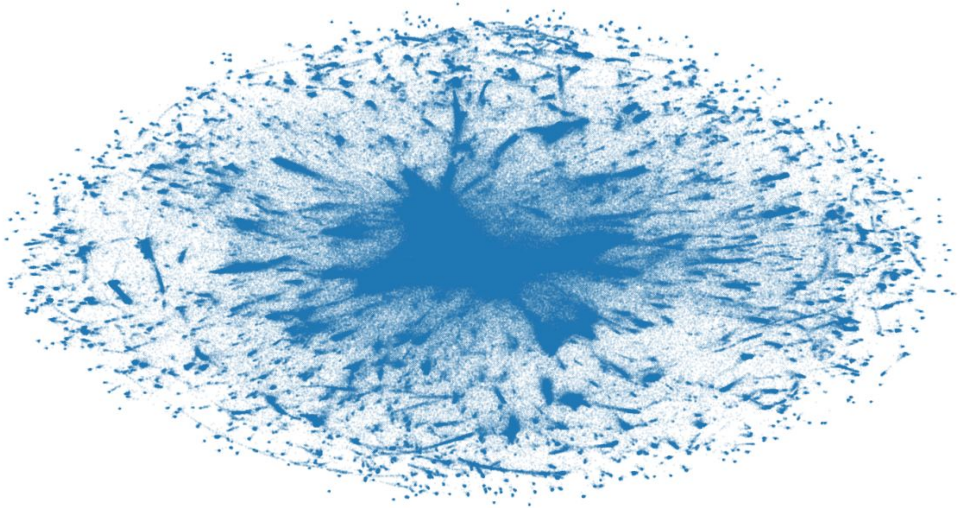


ResNet-200 Codes
(486 s)

VGG-19 Codes
(523 s)

- ResNet-200 demonstrates a more continuous space for latent codes
- Suggests a more continuous classification space with ResNet-200 than VGG-19

Applications - GLoVe Vectors



GLoVE Vectors
(2.2M Words, 300-dimensions, 573.2s)

- Hamming Distances play a significant role in the clusters. (i.e. clusters are frequently textually similar data, e.g. french words, dates)
- L2 metric could be questionable choice for comparing GLoVe vectors for certain instances.

Future Work

- Support multi-GPU (most requested feature on GitHub)
- Explore Fast Multipole Method/Fourier Methods to compute repulsive forces
- Explore improvements to the sparse matrix multiplication for computing attractive forces
- Explore interactive visualization for training machine learning models

Questions?

https://github.com/CannyLab/tsne-cuda