Alpha-Beta Divergences Discover Micro and Macro Structures in Data

Karthik Narayan KARTHIK.NARAYAN@BERKELEY.EDU
University of California, Berkeley, CA, 94720, USA

Ali Punjani ALIPUNJANI@CS.TORONTO.EDU
University of Toronto, ON M5S, CANADA

Pieter Abbeel PABBEEL@CS.BERKELEY.EDU
University of California, Berkeley, CA, 94720, USA

Abstract

Although recent work in non-linear dimensionality reduction investigates multiple choices of divergence measure during optimization (Yang et al., 2013; Bunte et al., 2012), little work discusses the direct effects that divergence measures have on visualization. We study this relationship, theoretically and through an empirical analysis over 10 datasets. Our works shows how the α and β parameters of the generalized alpha-beta divergence can be chosen to discover hidden macro-structures (categories, e.g. birds) or micro-structures (fine-grained classes, e.g. toucans). Our method, which generalizes t-SNE (van der Maaten, 2008), allows us to discover such structure without extensive grid searches over (α, β) due to our theoretical analysis: such structure is apparent with particular choices of (α, β) that generalize across datasets. We also discuss efficient parallel CPU and GPU schemes which are non-trivial due to the tree-structures employed in optimization and the large datasets that do not fully fit into GPU memory. Our method runs 20x faster than the fastest published code (Vladymyrov & Carreira-Perpinán, 2014). We conclude with detailed case studies on the following very large datasets: ILSVRC 2012, a standard computer vision dataset with 1.2M images; SUSY, a particle physics dataset with 5M instances; and HIGGS, another particle physics dataset with 11M instances. This represents the largest published visualization attained by SNE methods. We have open-sourced our visualization code: `http://rll.berkeley.edu/absne/`.

Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37. Copyright 2015 by the author(s).

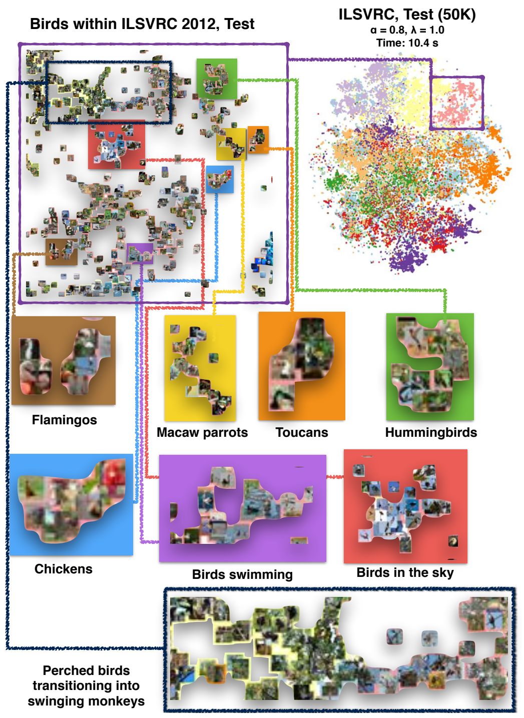

Figure 1. Discovering micro-structures using the settings $\alpha < 1, \lambda = 1$, prescribed by the intuition in (Section 4). Using Caffe fc7 features (Jia et al., 2014), we discover structures in the ILSVRC 2012 test set, e.g., classes of birds, and visual transitions, e.g., perched birds transitioning smoothly to swinging monkeys.

1. Introduction and Related Work

Data visualization techniques aim to generate a low-dimensional representation of a dataset which data scientists and researchers can inspect to gain insight into the structure and complexity of the data. Vital to data-driven decision making, visualizations graphically represent latent structure and meaning in the dataset. Many recent approaches produce low-dimensional embeddings of data instances, which we can view via scatter plots (Carreira-Perpinan, 2010; Hinton & Roweis, 2003; van der Maaten, 2008; Lawrence, 2011; Bunte et al., 2012; Vladymyrov & Carreira-Perpinán, 2014; Yang et al., 2014). Particularly, recent variants on *stochastic neighborhood embedding* (SNE) have become popular (Hinton & Roweis, 2003); the recently published t-SNE (t-distributed stochastic neighbor embedding) is particularly popular (van der Maaten, 2008). At a high level, SNEs (i) capture neighborhood information from pairs of points in the original dataset with m data instances into an $m \times m$ data similarity matrix, and (ii) learn a low-dimensional embedding of those points whose similarity matrix closely matches the original. Computing an embedding which matches the structure of the original data involves minimizing a divergence measure, e.g., KL-divergence, between the data and embedding similarity matrices. Indeed, most methods employ the KL-divergence (Hinton & Roweis, 2003; van der Maaten, 2008; 2013; Yang et al., 2009).

Unfortunately, most proposed methods either (1) are hyperparameter-free, giving researchers little room to directly communicate what types of patterns they are searching for in the data, or (2) have non-intuitive hyperparameters that require expensive, tedious grid searches.

Contributions. We propose a method featuring 2 hyperparameters (α, β). Our theoretical analysis predicts that (1) setting $\alpha + \beta < 1$ reveals macro-structures (categories, e.g., dogs), (2) $\alpha < 1$ reveals micro-structures (fine-grained classes, e.g. dalmations), and (3) $\alpha + \beta > 1$ reveals instances close to class boundaries (e.g., digits that are easily confused as 1 vs 7 or 4 vs 9) (Section 4). The meaning of the parameters makes data exploration intuitive, and can obviate the need for extensive grid searches over hyperparameter settings. We emphasize that these settings are *dataset-agnostic*, empirically substantiated on 10 datasets covering a wide swath of sizes (100 – 11M instances) drawn from a broad set of domains (computer vision, biology, particle physics). Our method allows for fast parallel CPU/GPU implementations; our GPU implementation runs 20× faster than the state of the art SNE-based implementations (Section 5).

Our theoretical analysis additionally answers a question recently posed in (Bunte et al., 2012): *under what circumstances should we choose a particular divergence to minimize in the SNE framework, and what consequences does this choice have?* While minimizing divergences other than the KL-divergence in the SNE objective has recently been explored computationally, these works do not answer this question. Yang et. al. demonstrate that several popular SNE variants arise from varying the divergence that is being minimized (Yang et al., 2013). Yang et. al. show that a novel optimization equivalence theorem between α-divergences, β-divergences, and γ-divergences yields a class of methods that build on the best aspects of graph layout and vectorial embedding. Bunte et. al. apply t-SNE variants using several Bregman divergences, f-divergences, and γ-divergences to two small datasets (COIL-20 (Nene et al., 1996) and the Olivetti face dataset (Samaria & Harter, 1994)) (Bunte et al., 2012). Although Bunte et. al. acknowledge that varying divergences produce different visualizations, they admit that they are unable to deliver an overall recipe for choosing a particular divergence in a given task. To the best of our knowledge, this paper is the first to provide such a recipe.

Past work primarily explores minimizing purely a single divergence in the SNE framework. We discover that minimizing the generalized alpha-beta divergence (a.k.a. AB-divergence) (Cichocki et al., 2011), which blends the α- and β- divergences, is crucial to discovering important micro and macro structures in the data (Section 4).

2. Background: t-distributed Stochastic Neighborhood Embedding

Given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m\}$, with data instances $\mathbf{x}_i \in \mathbb{R}^n$, t-distributed Stochastic Neighbor Embedding (t-SNE) (van der Maaten, 2008) aims to learn an embedding $\mathcal{E} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_m\}$, where $\mathbf{y}_i \in \mathbb{R}^d$ (usually, $d = 2$ or 3). To achieve this goal, t-SNE defines

$$\mathbf{P}_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{ik}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}, \tag{1}$$

$$\mathbf{P}_{ij} = (\mathbf{P}_{i|j} + \mathbf{P}_{j|i})/2m, \tag{2}$$

$$\mathbf{Q}_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}} \tag{3}$$

where additionally, $\mathbf{P}_{i|i} = \mathbf{Q}_{ii} = 0$. Determining the individual variances σ_i^2 involves running a binary search such that the *perplexity* (2 raised to the entropy) of the conditional $\mathbf{P}_{\cdot|j}$ equals k, a free parameter (van der Maaten, 2013). The embedding employs a Student-t kernel rather than a Gaussian to prevent embedding points from crowding near the center of the visualization map without clear clustering, a.k.a. the "crowding problem." t-SNE prescribes learning the embedding vectors $\mathbf{y}_i$ by running gradient descent to minimize the resulting non-convex KL-divergence, $\mathcal{J}(\mathcal{E}) = \sum_{i \neq j} \mathbf{P}_{ij} \log \mathbf{P}_{ij}/\mathbf{Q}_{ij}$:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{y}_i} = 4\sum_{i \neq j} Z(\mathbf{y}_i - \mathbf{y}_j)(\mathbf{P}_{ij}\mathbf{Q}_{ij} - \mathbf{Q}_{ij}^2) \tag{4}$$

$$= \sum_{i \neq j} \underbrace{4\mathbf{P}_{ij}\mathbf{Q}_{ij}Z(\mathbf{y}_i - \mathbf{y}_j)}_{\text{Force 1}} - \sum_{i \neq j} \underbrace{4\mathbf{Q}_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)}_{\text{Force 2}} \tag{5}$$

where $Z = \sum_{k \neq l}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}$. Naively computing this gradient takes $\mathcal{O}(m^2)$ time, making visualizations of greater than 50K data vectors prohibitively expensive.

Barnes-Hut-SNE (BHSNE) (van der Maaten, 2013) approximates the t-SNE's gradient in sub-quadratic time, yielding nearly identical visualizations to t-SNE while allowing for visualizations of millions of data vectors in a few hours. To this end, BHSNE only retains $\mathbf{P}_{ij}$ where data vector $\mathbf{x}_j$ is one of $\mathbf{x}_i$'s closest $3k$ neighbors, and sets the rest to 0. In practice, BHSNE constructs a vantage point tree to execute all nearest neighbor searches in $\mathcal{O}(kmn \log m)$ time (van der Maaten, 2013; Yianilos, 1993). BHSNE computes Force 1 in $\mathcal{O}(km)$ time by adding only terms involving positive $\mathbf{P}_{ij}$ and computing each $\mathbf{Q}_{ij}Z = (1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}$ in constant time; we ignore the dimensionality of the $\mathbf{y}_i$, as BHSNE typically seeks a 2D or 3D embedding. BHSNE employs a Barnes-Hut approximation algorithm to compute Force 2 in $\mathcal{O}(m \log m)$ time: given $\mathbf{y}_i, \mathbf{y}_j$, and $\mathbf{y}_k$ where $\|\mathbf{y}_i - \mathbf{y}_j\| \approx \|\mathbf{y}_i - \mathbf{y}_k\| \gg \|\mathbf{y}_j - \mathbf{y}_k\|$, the contributions of $\mathbf{y}_j$ and $\mathbf{y}_k$ to Force 2 will be roughly equal. The Barnes-Hut algorithm exploits this in computing the sum of all contributions to an embedding vector $\mathbf{y}_i$ by (i) constructing a quadtree over $\{\mathbf{y}_i\}$, (ii) traversing the quadtree via a depth-first-search, and (3) at every quadtree node, deciding whether the corresponding cell can summarize the gradient contributions for all points in that cell. In computing Force 2 for a point $\mathbf{y}_i$, if a cell is sufficiently small and far away from $\mathbf{y}_i$, then $\mathbf{Q}_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)$ will be similar for all points $\mathbf{y}_j$ in that cell. As such, BHSNE approximates the total contribution as $N_{cell} \cdot \mathbf{Q}_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)$ where N_{cell} denotes the total number of points in the cell. To compute Z efficiently, BHSNE (i) runs a separate Barnes-Hut procedure to compute a $z_i = \sum_j K_q(\|\mathbf{y}_i - \mathbf{y}_j\|^2)$ for each embedding point i and (ii) sums over the z_i's to yield Z. BHSNE then uses this value of Z in the Barnes-Hut procedure to compute Force 2. BHSNE decides whether a cell can summarize the points that it contains by checking whether $\|\mathbf{y}_i - \mathbf{y}_{cell}\|^2 / r_{cell} < \theta$, where r_{cell} is the length of the cell diagonal, y_{cell} is the cell's center of mass, and θ is a threshold that trades off speed and accuracy (larger values lead to poorer approximations).

3. Alpha-Beta Stochastic Neighborhood Embedding

Alpha-Beta Stochastic Neighborhood Embedding (AB-SNE), our proposed method, differs from t-SNE in that it minimizes the alpha-beta divergence (AB-divergence). We minimize the cost $\mathcal{J}_{\text{ABSNE}}(\mathcal{E}; \alpha, \beta) = D_{AB}^{\alpha\beta}(\mathbf{P}\|\mathbf{Q})$, computed as

$$\frac{1}{\alpha\beta} \sum_{i \neq j} \left(-\mathbf{P}_{ij}^{\alpha} \mathbf{Q}_{ij}^{\beta} + \frac{\alpha}{\alpha+\beta} \mathbf{P}_{ij}^{\alpha+\beta} + \frac{\beta}{\alpha+\beta} \mathbf{Q}_{ij}^{\alpha+\beta} \right), \tag{6}$$

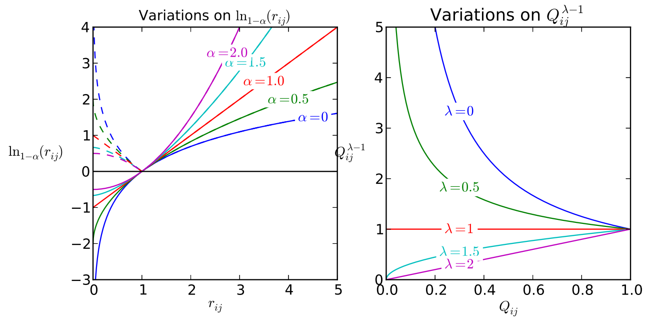

Figure 2. Functions in the (left) Tsallis deformed q-logarithm and (right) power families. Dotted lines in the left figure denote absolute values of functions. $|\ln_{1-\alpha}(r_{ij})|$.

where $\alpha \in \mathbb{R} \setminus \{0\}, \beta \in \mathbb{R}$ are hyperparameters. It is possible to set $\beta = 0$ or $\alpha + \beta = 0$ by extending the AB-divergence via continuity, e.g., by applying l'Hôpital's rule (see (Cichocki et al., 2011)); this does not affect the form of the gradient we present below (see Supplementary Materials). We use the definitions of $\mathbf{P}_{ij}$ and $\mathbf{Q}_{ij}$ employed in BHSNE (see Section 2). We minimize the ABSNE objective via gradient descent. The gradient, $\partial \mathcal{J}_{\text{ABSNE}} / \partial \mathbf{y}_i$, is computed as

$$\sum_j 4Z\mathbf{Q}_{ij}^2(\mathbf{y}_i - \mathbf{y}_j)(\underbrace{\mathbf{P}_{ij}^{\alpha}\mathbf{Q}_{ij}^{\beta-1}}_{\text{Force 1}} - \underbrace{\mathbf{Q}_{ij}^{\alpha+\beta-1} - J_1 + J_2)}_{\text{Force 2}} \tag{7}$$

where $J_1 = \sum_{k \neq l} \mathbf{P}_{kl}^{\alpha} \mathbf{Q}_{kl}^{\beta}$ and $J_2 = \sum_{k \neq l} \mathbf{Q}_{kl}^{\alpha+\beta}$. We compute ABSNE gradients in $\mathcal{O}(m \log m + mk)$ time using BHSNE's computational tricks: we can compute Force 1 and J_1 in $\mathcal{O}(km)$ time using $\mathbf{P}$'s sparsity and Force 2 via a Barnes-Hut algorithm similar to the one described in Section 2 after pre-computing Z and J_2 using a separate Barnes-Hut procedure.

4. How α and β Discover Hidden Structures

The AB-divergence offers two hyperparameters, α and β, which have strong intuitive meaning. This conveniently removes the need for tedious grid searches. Defining $\lambda = \alpha + \beta$, we inspect the updates taken during learning:

$$\Delta \mathbf{y}_i = -\frac{\partial D_{AB}^{\alpha\beta}(\mathbf{P}\|\mathbf{Q})}{\partial \mathbf{y}_i} = \sum_j \frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i} \mathbf{Q}_{ij}^{\lambda-1} \ln_{1-\alpha}\left(\frac{\mathbf{P}_{ij}}{\mathbf{Q}_{ij}}\right) \tag{8}$$

$$= \sum_j \frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i} \mathbf{Q}_{ij}^{\lambda-1} \ln_{1-\alpha}(r_{ij}) \tag{9}$$

where $\ln_q(x)$ is the Tsallis deformed q-logarithm[1] and $r_{ij} = \mathbf{P}_{ij}/\mathbf{Q}_{ij}$. Our theoretical analysis considers how

[1] $\ln_q(x) \equiv (x^{1-q} - 1)/(1-q)$ if $q \neq 1$, else $\ln_q(x) = \ln x$.

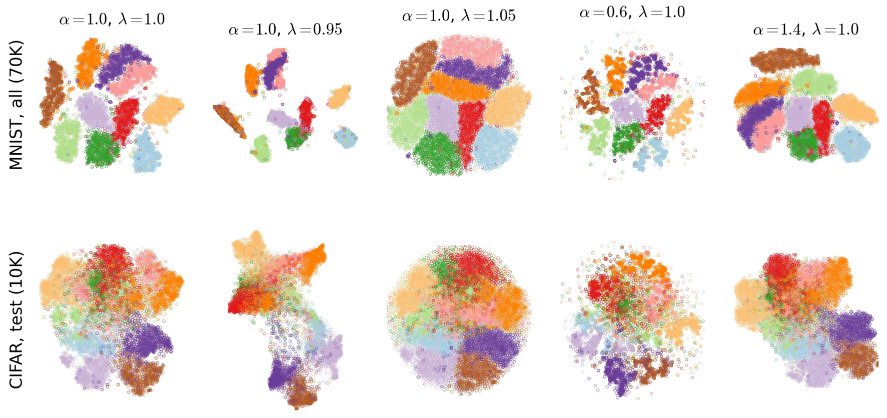

Figure 3. Empirical evidence of the theory in Section 4: (col. 2) $\lambda < 1$ reveals macro-structures, (col. 4) $\alpha < 1$ reveals micro-structures, and (col. 3) $\lambda > 1$ reveals instances close to class boundaries (Section 4). Further evidence on larger datasets (1M+ instances) is provided in Section 6. During data analysis, if supervision is not provided, column 2 may help in identifying classes. If supervision is provided, the right three plots can help understand "easily confused" (boundary) instances and instances within fine-grained categories. For reference, the left-most column displays t-SNE's visualization, i.e., in the limit $(\alpha, \lambda) \to (1, 1)$ (this derivation is non-trivial due to the limits and presented in the Supplementary Materials).

each force $\mathbf{f}_{ij} = \partial \mathbf{Q}_{ij}/\partial \mathbf{y}_i \cdot \mathbf{Q}_{ij}^{\lambda-1} \ln_{1-\alpha}(r_{ij})$ affects $\Delta \mathbf{y}_i$. The term $\partial \mathbf{Q}_{ij}/\partial \mathbf{y}_i$ is a vector parallel to the ray $k(\mathbf{y}_i - \mathbf{y}_j)$. Since the t-distribution monotonically decreases for positive arguments, we must have $k < 0$, i.e., $\mathbf{f}_{ij}$ points *towards* $\mathbf{y}_j$, implying that $\mathbf{f}_{ij}$ attracts $\mathbf{y}_i$ towards $\mathbf{y}_j$ if $\mathbf{Q}_{ij}^{\lambda-1} \ln_{1-\alpha}(r_{ij}) > 0 \Rightarrow r_{ij} > 1$ (see Figure 2) and repulses $\mathbf{y}_i$ from $\mathbf{y}_j$ if $r_{ij} < 1$. Specifically, non-neighboring point pairs in the original dataset $\mathcal{D}$ with $\mathbf{P}_{ij} = 0 \Rightarrow r_{ij} = 0 < 1$ repel each other.

We interleave theoretical intuition with empirical verification on two datasets (more results presented in Section 6): MNIST (LeCun & Cortes, 1998), a dataset of 70K 28×28 grayscale images depicting handwritten digits 0-9 and CIFAR-10 (Krizhevsky & Hinton, 2009), a dataset of 32×32 color images depicting 10 distinct object classes. We use the raw pixel data as MNIST's feature representation. For CIFAR-10, we train a 3-layer convolutional neural network with the CUDACONVNET (Krizhevsky, 2011) architecture using Caffe (Jia et al., 2014) and employ only third layer convolutional features; we visualize the test set to avoid having training labels directly influence the embedding. Applying PCA to center and reduce each dataset to 100 dimensions, we ran ABSNE under various (α, λ) for both datasets with perplexity 30 (Figure 3). We initialize all $\mathbf{y}_i$ in experiments with the same random seed and run exactly 1000 iterations of gradient descent per configuration (see Section 5 for optimization details), so structural details across a row should be comparable; the first column ($\alpha = 1.0, \lambda = 1.0$) denotes vanilla t-SNE.

Intuition Behind α. To study α, let us fix $\lambda = 1$. Consider a cluster of embedding points consisting of a few sub-clusters. After convergence, the sub-clusters will be placed together in such a way that the attractive and repulsive forces are balanced. According to Figure 2a., decreasing α below 1 emphasizes the magnitude of (repulsive) forces with $r_{ij} < 1$ relative to (attractive) forces with $r_{ij} > 1$. The emphasized repulsive forces and diminished attractive forces should cause sub-clusters to be placed further apart, implying that *ABSNE should tend to produce lots of small, fine-grained clusters for $\alpha < 1$.* Because $r_{ij} < 1 \Rightarrow \mathbf{Q}_{ij} > \mathbf{P}_{ij}$ implies that points $\mathbf{y}_i, \mathbf{y}_j$ are closer than they are supposed to be, the $\mathbf{f}_{ij}$ operating on close-proximity points $\mathbf{y}_i, \mathbf{y}_j$ should be emphasized more than those of far away points, implying that *setting $\alpha < 1$ should lead to fewer global changes in visualization structure in comparison with t-SNE ($\alpha = \lambda = 1$), but lots of change in local structure.* Similarly, *setting $\alpha > 1$ should lead to fewer, larger clusters with more global visualization changes.* Inspecting columns 4 and 5 in Figure 3, we notice that varying α yields the anticipated effect of local clustering: in both CIFAR-10 and MNIST, individual clusters are more tight and fine-grained for $\alpha < 1$ and loose for $\alpha > 1$. As predicted, little global restructuring takes place for $\alpha < 1$ in comparison with $\alpha > 1$. Variations on α could be useful to a user interested in inspecting the relationships between sub-clusters arising within larger clusters of the data at various scales.

Intuition Behind λ. To study λ, let us fix $\alpha = 1$. According to Figure 2b., setting $\lambda < 1$ emphasizes $\mathbf{f}_{ij}$ with low $\mathbf{Q}_{ij}$ (distant $\mathbf{y}_i, \mathbf{y}_j$), over $\mathbf{f}_{ij}$ with high $\mathbf{Q}_{ij}$ (near $\mathbf{y}_i, \mathbf{y}_j$). So, *changing λ should primarily affect global over local structure.* Thus, $\lambda < 1$ increases the magnitudes of forces on distant, repulsive point pairs, exaggerating repulsion of non-neighboring point pairs in the original dataset, *which* *we conjecture leads to greater cluster separation while setting $\lambda > 1$ leads to low separation.* Inspecting column 2, setting $\lambda < 1$ yields the anticipated effect of greater cluster separation: examining MNIST for $\lambda = 0.95 < 1$, ABSNE places each cluster of points further away from the others. Similarly, the purple and brown clusters are more separated from the other clusters with CIFAR-10. In column 3, setting $\lambda = 1.05 > 1$ yields a single large glob of points containing smaller globs corresponding to the same class, as expected. This setting could be useful if the user wishes to inspect "boundary" cases between embedding points with known classes.

The Importance of Blending α and β. While past work has individually applied the α- and β- divergences to the SNE problem (Yang et al., 2014; Bunte et al., 2012; Yang et al., 2013), the heavy dependence of the theory on $\lambda = \alpha + \beta$ shows that incorporating both divergences in the objective is crucial to discovering important micro and macro structures in data.

5. Parallel CPU and GPU Implementations

While many optimization methods exist for embeddings, e.g. spectral descent (Memisevic & Hinton, 2005), partial Hessian strategies (Vladymyrov & Carreira-Perpinan, 2012), fast multipole methods with L-BGFGS (Vladymyrov & Carreira-Perpinán, 2014), we found that warm-started gradient descent (van der Maaten, 2013) obtained strong results: we (1) initialize all $\mathbf{y}_i$ from a 2D isotropic Gaussian with variance 10^{-4} and (2) update each $\mathbf{y}_i$ via gradient descent (GD) with momentum (step size 200). For the first 250 descent iterations, we use momentum 0.5 and multiply all $\mathbf{P}_{ij}$ values by a user-defined constant $\alpha = 12$. For the last 750 iterations, we use momentum 0.8. We use a per-parameter adaptive learning rate scheme to speed up GD convergence (Jacobs, 1988), otherwise known to be very slow and sensitive to local optima in practice (Vladymyrov & Carreira-Perpinan, 2012). Concrete reasonings behind these choices can be found in (van der Maaten, 2008; 2013). We now detail how to compute gradients and update the $\mathbf{y}_i$ on the GPU, particularly using the NVIDIA compute unified device architecture (CUDA).

5.1. Parallel GPU Gradients

We store the $\mathbf{y}_i$ in two arrays on the GPU, one array per dimension, to take advantage of cache locality, as done in (Burtscher & Pingali, 2011). We store the (sparse) affinity matrix $\mathbf{P}$ as a list of triplets. We take advantage of $\mathbf{P}$'s symmetry to minimize the number of memory reads by storing only the upper half of the matrix. Our implementation consists of 13 kernels, which we now discuss.

Kernels 1 – 5: Partially Computing Force 2. Recall from Section 3 that the ABSNE gradient consists of two types of forces: Force 1 and Force 2. We first compute Force 2, since it will yield structures useful in computing Force 1.

Discussed in Section 2, computing Force 2 involves building a quadtree over the $\mathbf{y}_i$. To do this, we (Kernel 1) construct a bounding box over the $\mathbf{y}_i$, (Kernel 2) hierarchically subdivide the bounding box until each cell contains at most a single $\mathbf{y}_i$, and (Kernel 3) compute the center of mass and cumulative mass per cell. Next, we (Kernel 4) perform an in-order traversal of the quadtree, which approximately places nearby cells next to each other in the traversal; this is crucial in accelerating Kernel 5, which actually computes the forces on the individual $\mathbf{y}_i$.

To understand why the in-order traversal is necessary, recall that in CUDA, all threads within a single warp will execute in lockstep on entering a conditional statement only if the conditional evaluation is identical for all threads; otherwise, threads within the warp belonging to different branches will execute serially, often severely affecting performance. The in-order traversal substantially reduces such within-warp thread divergence in Kernel 5, leading to an order of magnitude savings in run-time. For more details, see (Burtscher & Pingali, 2011), which we follow closely in implementing these five kernels.

Jointly, this set of kernels computes and stores $g_1 = \sum_j 4Z\mathbf{Q}_{ij}^2(\mathbf{y}_i - \mathbf{y}_j)$ and $g_2 = \sum_j 4Z\mathbf{Q}_{ij}^2(\mathbf{y}_i - \mathbf{y}_j)\mathbf{Q}_{ij}^{\alpha+\beta-1}$ in GPU memory. We incorporate the contributions of J_1 and J_2 in later kernels.

Kernel 6, 7: Computing J_2 and Z. We evaluate $J_2 = \sum_{k \neq l} \mathbf{Q}_{kl}^{\alpha+\beta}$ by (1) computing a $J_2^{(i)} = \sum_{k \neq i} \mathbf{Q}_{ki}^{\alpha+\beta}$ per $\mathbf{y}_i$ and (2) executing a reduction to compute $J_2 = \sum_i J_2^{(i)}$. We efficiently perform (1) by computing auxiliary $J_2^{(i)}$ variables per $\mathbf{y}_i$ in executing Kernel 5. We perform the reduction through the open-source Thrust library (Bell & Hoberock, 2011). We compute Z similarly.

Kernel 8, 9: Computing J_1, Partially Computing Force 1. Rather than computing Force 1 by iterating over each $\mathbf{y}_i$, we iterate through the positive entries of $\mathbf{P}$, whose contributions we add to the prescribed $\mathbf{y}_i$ (Kernel 8). To parallelize computation, we split the list of triplets $(i, j, \mathbf{P}_{ij})$ across enough blocks to allow for 1024 threads per block; each thread processes a single triplet and updates points i and j. One caveat of this approach is that these updates must be made atomically; e.g., parallel updates for tuples $(2, 3)$ and $(3, 5)$ without atomic updates would yield a race condition for $\mathbf{y}_3$. Kernel 8 computes and stores $g_3 = \sum_j 4Z\mathbf{Q}_{ij}^2(\mathbf{y}_i - \mathbf{y}_j)\mathbf{P}_{ij}^{\alpha}\mathbf{Q}_{ij}^{\beta-1}$ in GPU memory.

Computing J_1 entails iterating through the positive entries of $\mathbf{P}$, computing the associated summands, and executing a reduce operation (Kernel 9). With large datasets, the GPU cannot fully store $\mathbf{P}$ in memory; so, we swap batches of $\mathbf{P}$ between CPU and GPU memory and apply Kernels 8 and

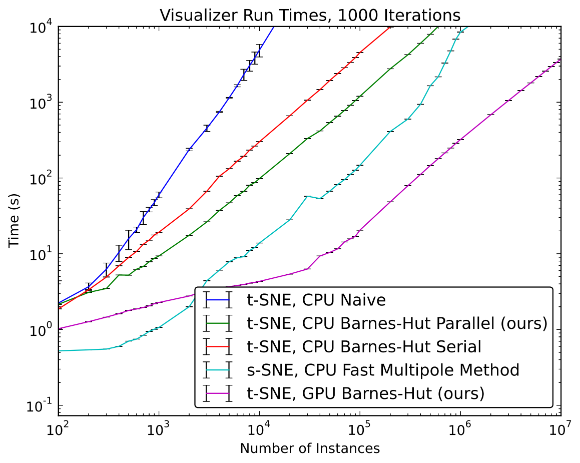

Figure 4. Best viewed in color. Timing experiments comparing CPU-NAIVE (van der Maaten, 2008), FMM (Vladymyrov & Carreira-Perpinán, 2014), CPU-BH (van der Maaten, 2013) with our CPU-PAR-BH and GPU-BH implementations. Error bars denote 2 standard deviations of time across 30 experiments.

9 per batch, accumulating only J_1 and g_3 in GPU memory.

Kernels 10 – 12: Updating $\mathbf{y}_i$. Kernel 10 executes the gradient updates per $\mathbf{y}_i$. We compute the gradient by modifying each $\mathbf{y}_i$ according to the entry in $g = g_1 * (J_2 - J_1) - g_2 + g_3$, taking into account momentum and the adaptive learning rates described in Section 5.

5.2. Parallel CPU Gradients

We use a very similar computation flow in computing parallel CPU gradients. In computing Force 2 on the CPU, we build the quadtree serially, as this step typically takes less than 10% of the full training time. After constructing the quadtree, we similarly partially compute Force 2 using OpenMP to parallelize computations over each $\mathbf{y}_i$. We then compute Z and J_2 via OpenMP's parallel reduce operation, using similar auxiliary variables as in the GPU implementation. We compute Force 1 and J_1 serially; in practice, we found that using atomic additions in parallel ran slower. We then similarly update the $\mathbf{y}_i$ in parallel.

5.3. Performance Experiments

All experiments in this paper employ a machine with 2x Intel Xeon X5570 CPUs (8 cores total, 2.93 GHz), 64GB memory, and an NVIDIA Tesla K40c graphics card. Corroborated by (van der Maaten, 2013), fixing $\theta = 0.25$ offers a good tradeoff between speed and accuracy. We explore the efficiency and scalability of (1) CPU-NAIVE: a naive CPU implementation (van der Maaten, 2008), (2) CPU-BH: a serial Barnes-Hut CPU implementation (van der Maaten, 2013), (3) FMM: a fast multipole method, a scheme with linear gradient time complexity, the fastest published code available online) (Vladymyrov & Carreira-Perpinán, 2014), (4) CPU-PAR-BH: our parallel Barnes-Hut CPU implementation, and (5) GPU-BH: our Barnes-Hut GPU implementation. All implementations run t-SNE by setting $\alpha = 1, \beta = 0$. FMM runs s-SNE, since code for t-SNE was not available; timing comparisons are still valid, since the number of operations involved in computing s-SNE and t-SNE gradients are similar. In our experiments, we sample subsets of the HIGGS dataset (Baldi et al., 2014), a large dataset consisting of 11M instances and 28 features (see Section 6 for dataset details and visualizations); we use a perplexity of 20. Figure 4 summarizes our findings in a log-log plot. As expected, CPU-NAIVE timings have a slope of 2 while CPU-BH, GPU-BH, and FMM timings have slopes close to 1, indicating the expected theoretical complexities.

Barnes-Hut CPU Parallel On datasets with more than 50K instances, CPU-PAR-BH yields speedups of more than 3.5x over CPU-BH, approaching about half of linear speedup; we do not attain full linear speedup, since Force 1's computation is not parallelized. Corroborated by (Vladymyrov & Carreira-Perpinán, 2014), FMM runs substantially faster than CPU-BH ($20 - 30$x); on datasets with more than 1M instances, CPU-PAR-BH closes this gap to 2.5x.

Barnes-Hut GPU Parallel On datasets with more than 50K instances, GPU-BH yields speedups of $150 - 200$x over CPU-BH and $55 - 60$x over CPU-PAR-BH. While FMM is 2x faster than GPU-BH on datasets with size < 2K, GPU-BH is $5 - 10$x faster on datasets with size $20 - 500$K and more than 20x faster on datasets with size $1 - 10$M. Between 100 and 50K instances, GPU-BH has a slope smaller than 1; this happens because (i) Force 2 (quadtree) computations dominate computation time and (ii) we are able to process ALL data instances simultaneously on the GPU, effectively yielding $\mathcal{O}(\log m)$ computation time. At the 100K mark, Force 1 dominates computation times because (i) we need to update each data instance's gradient atomically while (ii) swapping portions of the sparse matrix P in and out of GPU memory, causing the slope to return to 1.

6. Case Studies

Evaluating visualization quality is a difficult problem; in previous work, researchers have used supervised labels in tandem with a k-nearest neighbors metric to quantitatively assess performance (van der Maaten, 2008; 2013; Yang et al., 2009). However, such scores may not directly translate to "better visualization": For example, a data-miner aspiring to explore micro, within-class clusters in a dataset without supervised labels may assign a low score to an embedding that perfectly separates high-level macro clusters. We now explore ABSNE for use in such *goal-driven* visualization. *We strongly encourage readers to use a computer in tandem with digital zoom set to around* 400% *in looking at the large visualization in this section.*

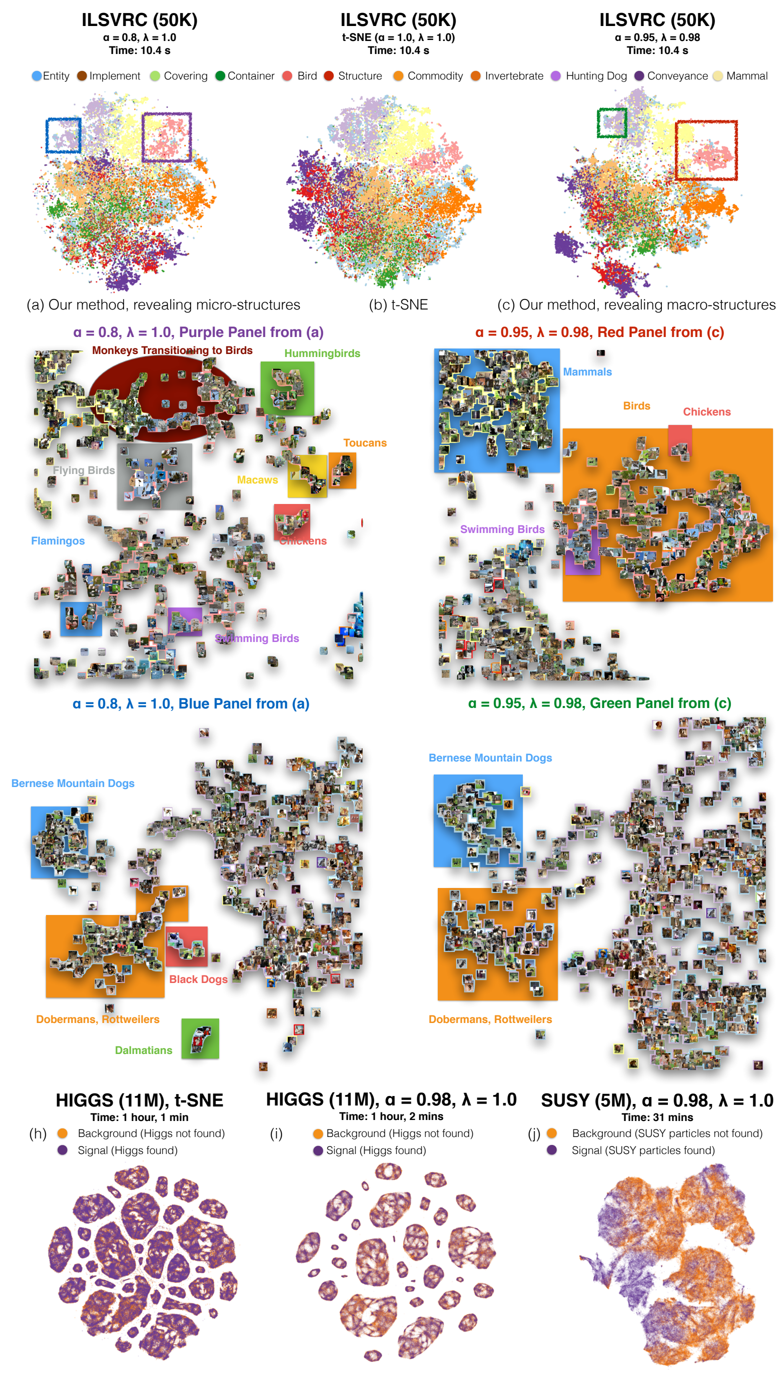

Figure 5. Please use digital zoom at high levels to view details. (a - c) depict ILSVRC 2012's micro (specific animal species) in the purple, blue panels and macro (classes under the animal kingdom) structures in the red, green panels. As expected, macro clusters are more widely separated in (c) than (a) and in the red, green panels than purple, blue panels. (h - j) depict analogous structures in the HIGGS and SUSY datasets; we are searching for what physical meaning these structures have in the Standard Model of particle physics.

6.1. ILSVRC 2012: Kingdoms to Species

A subset of ImageNet, ILSVRC 2012 (Deng et al., 2009) features a training set of 1.28M images of varying size, validation set of 50K images, and 1000 object categories. [2] We employ fc7 features yielded by Alexnet (Krizhevsky et al., 2012) implemented in Caffe (Jia et al., 2014), trained on the 1.28M images.[3] We show a t-SNE plot of the validation set in the top-middle of Figure 5; we also present plots and 2 zoom views for $(\alpha = 0.8, \lambda = 1)$ and $(\alpha = 0.95, \lambda = 0.98)$. Zoom views present a random subset of images from the plot; displayed images were not hand-picked. As Section 4 predicts, the top scatter plots show tight clusters and greater class separation in $(\alpha = 0.95, \lambda = 0.98)$ compared with the other settings, particularly clusters corresponding to birds, mammals, and dogs. The maroon oval in the purple panel shows a transition between perched birds to monkeys in trees, while the blue and orange boxes in the red panel show well-separated classes. As expected, we found stronger intra-class clustering for $(\alpha = 0.8, \lambda = 1)$; the purple panel shows separate clusters for hummingbirds (green), macaw parrots (yellow), toucans (orange), flamingos (blue), chickens (red), ducks (purple), and birds in the sky (gray). Similarly, the blue panel separately clusters dalmations (green), Bernese mountain dogs (blue), black and brown dogs, e.g. Doberman Pinschers and German Rottweilers (orange), and black dogs (red). In comparison, the red panel associated with $(\alpha = 0.95, \lambda = 0.98)$ shows fewer, vague clusters with all birds (orange): chickens (red) and swans/ducks (purple). The green panel shows fewer classes: Bernese mountain dogs (blue) and black and brown dogs (orange).

6.2. HIGGS: Discovering Higgs Bosons

We return to the HIGGS dataset (Baldi et al., 2014), whose goal is to distinguish between signal processes which produce Higgs bosons and background processes which do not. Each data instance consists of 28 features: the first 21 describe kinematic properties measured by detectors in the accelerator, while the last 7 are high-level functions of the first 21. Again, we first run t-SNE on the HIGGS dataset (bottom left of Figure 5). For HIGGS and SUSY, we first initialize and optimize with 0.5M random instances. Upon convergence, we place another 0.5M new random instances near their nearest neighbor in the original dataset with isotropic Gaussian noise with variance 0.01. Repeating until all points have been embedded, this produces final objectives with lower value.

Interestingly, while clustering occurs, the clusters don't correspond to the desired classes. Hoping that tighter global clustering will lead to more intuitive results, we set $\alpha = 0.98$ (bottom middle plot). As predicted in Section 4, straggling points align with existing clusters to yield a cleaner plot. While some clusters appear to have slightly denser concentrations of positive signals, there still is not any concrete class separation. While we are not aware of what the resulting clusters mean, we believe that the clusters could yield further insights.

6.3. SUSY: Discovering Supersymmetric Particles

Discovering evidence of supersymmetry (SUSY) constitutes a major goal in the Large Hadron Collider's central mission; one ramification of the theory includes the discovery of dark matter candidate particles (Baldi et al., 2014). We explore the SUSY dataset (Baldi et al., 2014), which similarly tries to distinguish between processes which do and do not produce supersymmetric particles. There is currently a vigorous effort to improve performance in this classification task (Cheng & Han, 2008; Barr et al., 2003; Rogan, 2010; Buckley et al., 2013). As with HIGGS, we found that setting $\alpha = 0.98, \lambda = 1$ yields a cleaner plot with greater separation than that of t-SNE (not shown due to lack of space). While we do not observe perfect cluster separation, there is a distinct purple region corresponding to observed SUSY particles. Perhaps, replicating experimental conditions leading to particles in this region would have a higher chance of yielding SUSY particles.

7. Discussion

Although several papers have explored varying divergences in SNE methods, this is the first paper to theoretically attribute and empirically verify how divergence parameters qualitatively affect visualization. Our analysis reveals that parameter variation in (α, β) for the AB-divergence discovers micro and macro structures within data in a dataset-agnostic fashion. Our well-optimized GPU implementation yields speedups of more than 20x on datasets with 1 – 10M instances over the state of the art implementation (Vladymyrov & Carreira-Perpinán, 2014). This yields the largest published SNE visualization of a dataset (11M instances).

8. Acknowledgements

We thank Eriz Tzeng, Animesh Garg, and Teodor Moldovan for useful discussions and NVIDIA for donating 2x Tesla K40 GPUs. KN was funded by an NSF Graduate Fellowship (Summer 2014) and by an NDSEG Fellowship (Fall 2014 - 2015). AP was partially funded by NSERC. This research was also funded in part by ONR through a Young Investigator Program award.

[2]These classes correspond to leaves in a hierarchical tree of classes, with "entity" being the root. For convenient visualization, we greedily group together classes with the same parent until 11 classes remain of roughly equal cardinality ((Donahue et al., 2013) employ a similar method).

[3]We visualize the validation set rather than the training set, because the training labels were used to train the Caffe model, and using the training labels themselves would yield perfect separation in visualization.

References

Baldi, P., Sadowski, P., and Whiteson, D. Deep learning in high-energy physics: improving the search for exotic particles. *arXiv preprint arXiv:1402.4735*, 2014.

Barr, A., Lester, C., and Stephens, P. mt2: the truth behind the glamour. *Journal of Physics G: Nuclear and Particle Physics*, 29(10):2343, 2003.

Bell, Nathan and Hoberock, Jared. Thrust: A productivity-oriented library for cuda. *GPU Computing Gems*, 7, 2011.

Buckley, M. R., Lykken, J. D., Rogan, C., and Spiropulu, M. Super-razor and searches for sleptons and charginos at the lhc. *arXiv preprint arXiv:1310.4827*, 2013.

Bunte, K., Haase, S., Biehl, M., and Villmann, T. Stochastic neighbor embedding (sne) for dimension reduction and visualization using arbitrary divergences. *Neurocomputing*, 90:23–45, 2012.

Burtscher, M. and Pingali, K. An efficient cuda implementation of the tree-based barnes hut n-body algorithm. *GPU Computing Gems Emerald edition*, pp. 75, 2011.

Carreira-Perpinan, M. A. The elastic embedding algorithm for dimensionality reduction. In *ICML*, 2010.

Cheng, H. and Han, Z. Minimal kinematic constraints and mt2. *Journal of High Energy Physics*, 2008(12):063, 2008.

Cichocki, A., Cruces, S., and Amari, S. Generalized alpha-beta divergences and their application to robust nonnegative matrix factorization. *Entropy*, 13:134–170, 2011.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

Hinton, G.E. and Roweis, S.T. Stochastic neighbor embedding. In *NIPS*, 2003.

Jacobs, R.A. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding, 2014.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Krizhevsky, Alex. cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks. `https://code.google.com/p/cuda-convnet/`, 2011.

Lawrence, N.D. Spectral dimensionality reduction via maximum entropy. In *AISTATS*, 2011.

LeCun, Y. and Cortes, C. The mnist database of handwritten digits, 1998.

Memisevic, R. and Hinton, G. Improving dimensionality reduction with spectral gradient descent. *Neural networks*, 18(5):702–710, 2005.

Nene, S.A., Nayar, S.K., and Murase, H. Columbia object image library (coil-20). Technical report, Technical Report CUCS-005-96, 1996.

Rogan, C. Kinematical variables towards new dynamics at the lhc. *arXiv preprint arXiv:1006.2727*, 2010.

Samaria, F. S. and Harter, A. C. Parameterisation of a stochastic model for human face identification. In *Applications of Computer Vision*, 1994.

van der Maaten, L.J.P. Visualizing high-dimensional data using t-sne. *JMLR*, 9:2579–2605, 2008.

van der Maaten, L.J.P. Barnes-hut-sne. In *ICLR*, 2013.

Vladymyrov, M. and Carreira-Perpinan, M. Partial-hessian strategies for fast learning of nonlinear embeddings. In *ICML*, 2012.

Vladymyrov, Max and Carreira-Perpinán, Miguel A. Linear-time training of nonlinear low-dimensional embeddings. In *AISTATS*, pp. 968–977, 2014.

Yang, Z., King, I., Xu, Z., and Oja, E. Heavy-tailed symmetric stochastic neighbor embedding. In *NIPS*, 2009.

Yang, Z., Peltonen, J., and Kaski, S. Scalable optimization of neighbor embedding for visualization. In *ICML*, 2013.

Yang, Zhirong, Peltonen, Jaakko, and Kaski, Samuel. Optimization equivalence of divergences improves neighbor embedding. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 460–468, 2014.

Yianilos, P.N. Data structures and algorithms for nearest neighbor search in general metric spaces. In *ACM-SIAM Symposium on Discrete Algorithms*, pp. 311–321, 1993.