

Hyperbolic Interpolation-Based t-SNE
Master's Thesis
TU Delft

Aaron Kavalier

August 2026

Hyperbolic Interpolation-Based t-SNE

Introducing a Novel Algorithm for Accelerating Hyperbolic t-SNE

by

Aaron Kavaler

to obtain the degree of Master of Science in Applied Mathematics at the Delft University of Technology, to be defended publicly on Monday August 31, 2026 at 15:00.

Student number: 6311148

Project duration: November 20, 2025 – August 31, 2026

Thesis committee:	Prof. Martin van Gijzen,	TU Delft Department of Applied Mathematics, Responsible Supervisor
	Prof. Martin Skrodzki,	TU Delft Computer Graphics and Visualization, Daily Supervisor
	Prof. David de Laat,	TU Delft Department of Applied Mathematics, Independent Committee Member

An electronic version of this thesis is available at
<http://repository.tudelft.nl/>.



Abstract

Hyperbolic t-SNE extends t-SNE to hyperbolic space, making it well suited for visualizing hierarchical datasets. However, its computational cost remains a barrier to scaling. This thesis introduces HIt-SNE, a method that accelerates hyperbolic t-SNE. It does so by adapting the interpolation-based strategy behind FIt-SNE to hyperbolic geometry. The approach reformulates the underlying gradient computation and partitions the embedding space using a hyperbolic tiling. Distant interactions are then approximated through local interpolation, while an exact correction term preserves accuracy for nearby points. We analyze the runtime and accuracy of the resulting algorithm and evaluate it on real-world datasets. We find that our algorithm substantially reduces computational cost relative to the exact method, while keeping approximation error within a controllable range. These results suggest that interpolation-based acceleration is a promising direction for making hyperbolic dimensionality reduction more scalable.

Contents

1	Introduction	4
2	Mathematical Background	6
2.1	Dim. Reduction	6
2.1.1	Linear Dimensionality Reduction Methods	7
2.1.2	Non-linear Dimensionality Reduction Methods	7
2.2	t-SNE	8
2.2.1	t-SNE Cost Function	9
2.2.2	Initialization and Gradient Descent	12
2.2.3	Early Exaggeration	12
2.3	Related work	13
2.3.1	Barnes-Hut acceleration	13
2.3.2	Other related methods	15
2.3.3	FIt-SNE	16
2.3.4	Use of the Fast Fourier Transform in FIt-SNE	18
2.4	Hyperbolic Space	20
2.4.1	Overview of Hyperbolic Space	20
2.4.2	Lorentz Transformations	22
2.5	Hyperbolic t-SNE	23
2.5.1	Basic method of hyperbolic t-SNE	23
2.5.2	Variations on hyperbolic t-SNE	25
2.5.3	Hyperbolic Barnes-Hut acceleration	25
2.6	RQ Refinement	27
3	HIt-SNE	29
3.1	Reformulating hyperbolic t-SNE	29
3.1.1	Hyperboloid formulation of hyperbolic t-SNE	29
3.1.2	Splitting the hyperbolic t-SNE gradient into two approxima- tion problems	30
3.1.3	Hyperbolic kernel sums	30
3.2	Local Interpolation	31
3.2.1	Tile mesh generation	31
3.2.2	Tile assignment	33
3.2.3	Local tangent coordinates	33
3.2.4	Interpolation grid and weights	34
3.2.5	Sparse-tile fallback	36
3.3	Efficient ϕ approximation	36
3.3.1	Far-field approximation	36

3.3.2	Near-field correction	37
3.4	Overview and analysis	38
3.4.1	Algorithm overview	39
3.4.2	Complexity	40
4	Experiments	43
4.1	Datasets	43
4.2	Setup	44
4.3	Per-iteration	45
4.4	Scaling	45
4.5	Visual Fidelity	46
5	Results	47
5.1	Per-iteration	47
5.2	Scaling	48
5.3	Visual Fidelity	49
6	Discussion	52
6.1	Runtime Behavior	52
6.2	Approximation Quality	52
6.3	Hybrid Method	53
6.4	Limitations	54
7	Conclusion	55
7.1	Answering the Research Question	55
7.2	Future Research Directions	56
A	Gradient Derivations	63
A.1	Generic Distance	63
A.1.1	Euclidean case	65
A.1.2	Poincaré disk	65
A.1.3	Hyperboloid model	66
A.2	Hyperboloid-Poincaré	67
B	HIt-SNE Derivations	71
B.1	HIt-SNE Kernel and Source Derivation	71
B.2	Error Bounds	71
B.2.1	Error Bound (FIt-SNE)	72
B.2.2	Error Bound (HIt-SNE)	73

Chapter 1

Introduction

Many modern datasets are both high-dimensional and hierarchical, yet many existing methods for analyzing such data fail to account for this structure. Data is considered hierarchical when the underlying structure resembles a tree, in the way that a biological taxonomy organizes species into orders, families, and genera, or the way a file system organizes files into nested folders. A defining feature of such hierarchical structures is that the number of nodes tends to grow exponentially with depth, since a tree where each node has b children has roughly b^d nodes at depth d . Many real-world high-dimensional datasets have this kind of latent hierarchical structure even when it is not given explicitly: word embeddings often encode an implicit hierarchy of semantic categories [13], single-cell RNA sequencing data reflects the hierarchical lineage of cell types [2] [22], and collections of scientific papers cluster into fields, subfields, and individual research topics [7].

One algorithm commonly used to understand high-dimensional data through visualization is t-SNE [31]. Like other dimensionality reduction algorithms, t-SNE works by constructing an *embedding*, a mapping from a high-dimensional dataset to a corresponding set of points in a much lower-dimensional space, usually two or three dimensions, chosen so that some meaningful property of the original data is preserved. For t-SNE specifically, the property preserved is local neighborhood structure, so that points which are close together in the high-dimensional data remain close together in the embedding. Once such an embedding has been constructed, it can be plotted as a scatterplot, making structure that is difficult to see directly in high dimensions, such as clusters of similar datapoints, visible to the eye.

However, the Euclidean embedding used by t-SNE does not naturally capture hierarchical structure. Euclidean volume grows only polynomially with radius, whereas, as noted above, the number of nodes in a tree grows exponentially with depth. This mismatch means that a tree embedded in Euclidean space will necessarily either crowd its nodes together or distort their relative distances. One solution to the problem of visualizing high-dimensional hierarchical data is to extend t-SNE to hyperbolic space. Hyperbolic space is an alternative to Euclidean space that can represent hierarchical structures, such as trees, with low distortion, since hyperbolic volume itself grows exponentially with radius [34], allowing tree-like structures to be embedded with much lower distortion than is possible in Euclidean space [41]. The extension of t-SNE to hyperbolic space is known as hyperbolic t-SNE, and several variants have been proposed in the literature [44] [20] [22] [49]. In this thesis we primarily focus on the approach of Skrodzki et al. [44].

Despite this advantage, hyperbolic t-SNE inherits the same computational bottleneck as ordinary t-SNE. Computing the repulsive force between embedding points requires summing over all $\binom{N}{2}$ pairs of points at every iteration of gradient descent, giving an $\mathcal{O}(N^2)$ cost per iteration. For Euclidean t-SNE, this bottleneck has been addressed by acceleration methods such as Barnes-Hut t-SNE and FIt-SNE, the latter of which approximates the repulsive force using polynomial interpolation, reducing the per-iteration cost to $\mathcal{O}(N)$ [27] [29]. Skrodzki et al. adapt the Barnes-Hut strategy to hyperbolic t-SNE [44], but no equivalent interpolation-based acceleration, of the kind used by FIt-SNE, has yet been developed for hyperbolic t-SNE. This represents a gap in the literature, since FIt-SNE is currently preferred over Barnes-Hut precisely because of its speed, and an extension of FIt-SNE to hyperbolic space therefore seems a natural next step.

This leads to the central research question of this thesis:

Can the interpolation-based acceleration strategy of FIt-SNE be extended to hyperbolic t-SNE?

We break this question down into three sub-questions which will be discussed in Section 2.6 after sufficient background information has been given.

We will answer our research question in the following chapters, organized as such. Chapter 2 provides the mathematical background necessary for understanding our proposed algorithm, including a brief overview of dimensionality reduction algorithms, t-SNE, hyperbolic geometry, and hyperbolic t-SNE. In Chapter 3, we describe our algorithm in detail. Chapter 4 lists the experiments we use to test the performance of our algorithm, and Chapter 5 presents their results, which are interpreted further in Chapter 6. The thesis concludes in Chapter 7 which summarizes our findings and gives ideas for future progress. The appendices include derivations of the parts of the algorithm and verification that our hyperboloid-model gradient agrees with the existing Poincaré-model formulation.

Chapter 2

Mathematical Background

2.1 Dimensionality Reduction

Dimensionality reduction is the process of taking a high-dimensional dataset and deriving a lower-dimensional dataset that has similar meaningful properties. Such properties may include: preservation of pairwise distances, nearest neighbors, or high-variance features.

In this thesis, we are mainly interested in the application of dimensionality reduction to visualizing the structure of high-dimensional data. Using a dimensionality reduction algorithm, a high-dimensional dataset can be reduced to two-or-three-dimensional space, where it can be visualized in a scatterplot. Some examples of this approach include understanding the approximately 30 dimensional space of parameters that describe breast cancer cells [31]. Another application is scientific literature exploration, where dimensionality reduction is used to make a map of similar papers by reducing the 768-dimensional space of paper description embeddings to 2D [44] [33] [11].

At a high level, dimensionality reduction procedures can be divided into linear and non-linear algorithms. Linear algorithms reduce the dimensionality of the data by embedding high-dimensional points into a lower-dimensional linear subspace. Dimensionality reduction can be highly effective if datapoints in high-dimensional space actually lie on a low-dimensional manifold. For datasets where the manifold that the data lies on is not a linear subspace, linear dimensionality reductions will not work, and non-linear methods are necessary (see Figure 2.1).

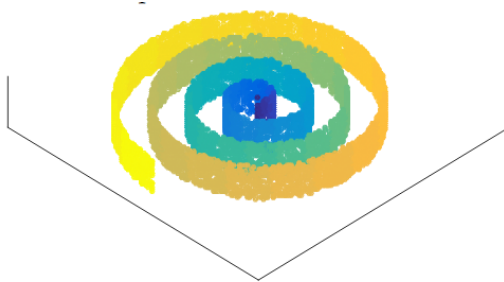


Figure 2.1: The swiss roll data set in $\mathbb{R}^3$. It is evident that the data lies completely on a 2D surface, which is not a linear subspace. In this case, non-linear dimensionality methods are needed to do dimensionality reduction while preserving the manifold's structure.

In the following section, we review linear and non-linear dimensionality reduction algorithms, in an effort to contextualize t-SNE. Linear algorithms include Principal Component Analysis (PCA) and classical multidimensional scaling (MDS) and will be reviewed in Section 2.1.1 [31]. Non-linear algorithms include Sammon Mapping, Isomap, Laplacian Eigenmaps (LE), Uniform Manifold Approximation and Projection (UMAP), and t-SNE variants (see Section 2.1.2). Since t-SNE and its variants will be the main focus of the thesis, we will discuss t-SNE in depth in Section 2.2.

2.1.1 Linear Dimensionality Reduction Methods

PCA is a linear dimensionality reduction technique which preserves the k directions of greatest variance in the n -dimensional data. Suppose a dataset consists of three columns, with variable data in the first two columns and near-constant values in the last column. Then PCA with $k = 2$ would preserve the span of the first two columns and filter out the near-constant last column. PCA can be calculated by computing a truncated Singular Value Decomposition (SVD) [45]. Specifically, first one subtracts the mean of each column from the column to create a centered data matrix where each column has mean zero [45]. Then one takes the SVD of this centered data matrix, and truncates the $n - k$ smallest singular values [45]. The result is a k -dimensional representation of the data obtained by projecting onto the subspace spanned by the top k singular vectors.

Multidimensional Scaling (MDS) is a dimensionality reduction technique that seeks a set of low-dimensional points whose pairwise distances closely match those of the original high-dimensional data [25]. Let x_i denote a datapoint and let $d_{ij} = \|x_i - x_j\|$ be the distance between points x_i and x_j . If y_i and y_j are the corresponding low-dimensional embeddings, then their distance is given by $\|y_i - y_j\|$. Let

$$\mathbf{Y} = \begin{bmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_N^T \end{bmatrix}$$

denote the matrix whose rows are the low-dimensional embeddings. MDS determines $\mathbf{Y}$ by minimizing a cost function, S , known as the *stress* [25]:

$$S(\mathbf{Y}) = \sum_{i < j} (d_{ij} - \|y_i - y_j\|)^2.$$

An interesting and surprising result is that the two most popular linear dimensionality reduction algorithms PCA and MDS are in fact mathematically identical [30]. This reveals that PCA, like MDS, is primarily driven by preserving large pairwise distances between datapoints, often at the cost of local structure. Because nearby points contribute comparatively little to the objective, local neighborhoods can be distorted in the embedding. This motivates nonlinear dimensionality reduction methods, which explicitly prioritize preserving local structure instead.

2.1.2 Non-linear Dimensionality Reduction Methods

Sammon mapping is a non-linear dimensionality reduction technique that amends classical MDS by introducing a distance-dependent weighting into the stress func-

tion, thereby placing greater emphasis on preserving small pairwise distances. The Sammon cost function is given by

$$\phi(\mathbf{Y}) = \frac{1}{\sum_{ij} d_{ij}} \sum_{i \neq j} \frac{(d_{ij} - \|y_i - y_j\|)^2}{d_{ij}},$$

which is typically minimized using pseudo-Newton methods [30].

Isomap is a non-linear dimensionality reduction technique which modifies MDS by using *geodesic distances* instead of Euclidean distances [46]. The geodesic distance $d_G(x_i, x_j)$ between datapoints x_i and x_j is computed as follows. First, the k -nearest neighbor graph with adjacency matrix $\mathbf{W}$ is constructed, where $W_{ij} = 1$ if x_i and x_j are k -nearest neighbors and 0 otherwise. The geodesic distance is then defined as the shortest path distance between x_i and x_j along the graph $\mathbf{W}$, with edge weights given by d_{ij} :

$$d_G(x_i, x_j) = \min_{p: x_i \rightarrow x_j} \sum_{(k,l) \in p} d_{kl},$$

where the minimum is taken over all paths p connecting x_i and x_j along edges of $\mathbf{W}$. In practice this is computed efficiently using a shortest-path algorithm such as Floyd–Warshall [46]. Using these geodesic distances in place of Euclidean distances, Isomap then proceeds as standard MDS, minimizing the stress function:

$$\phi(\mathbf{Y}) = \sum_{i < j} (d_G(x_i, x_j) - \|y_i - y_j\|)^2.$$

Isomap is often unreliable due to its strong assumption that the topology of the k -nearest neighbor graph resembles the topology of the manifold the data lies on [30].

Laplacian Eigenmaps (LE) is a non-linear dimensionality reduction method which uses a k -nearest neighbor graph $\mathbf{W}$ of the high-dimensional dataset to provide a filter to focus on local behaviour [4]. In particular, LE minimizes the following cost function:

$$\phi(\mathbf{Y}) = \sum_{ij} \|y_i - y_j\|^2 W_{ij},$$

which can be interpreted as minimizing the distance in the embedding between a datapoint’s k nearest neighbors [30]. The “Laplacian” in the name refers to the fact that this cost function can be rewritten as

$$\phi(\mathbf{Y}) = \text{tr}(\mathbf{Y}^T \mathbf{L} \mathbf{Y}),$$

where $\mathbf{L}$ is the graph Laplacian of $\mathbf{W}$. Justification for the use of the graph Laplacian, in terms of its relationship to the Laplace–Beltrami operator and the latter’s properties, is given in [4]. Additionally, “eigenmap” refers to the fact that the solution to the minimization of ϕ corresponds to the eigenvectors with smallest non-zero eigenvalues [4].

2.2 t-SNE

t-SNE is a non-linear gradient-based dimensionality reduction algorithm, and is popular for visualization as it produces an easily interpretable cluster structure (see

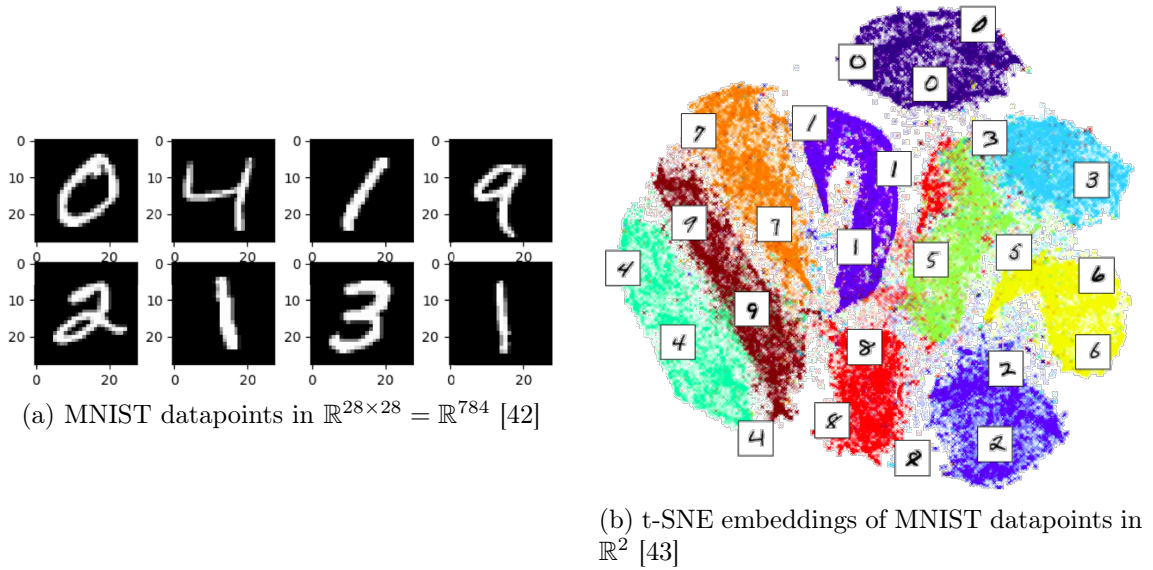


Figure 2.2: (a) The MNIST dataset consists of 10k handwritten grayscale digits and is commonly used as a test dataset in machine learning. (b) Using t-SNE to embed the datapoints into 2D, a scatterplot of the embeddings can be made and labels for the digits added. Each color corresponds to a digit 0-9, and strong clustering is evident.

Figure 2.2) [31]. t-SNE works by optimizing an information-theoretic cost function that will be described in Section 2.2.1. The cost function is optimized using gradient descent, and can be thought of as comprising three sequential phases:

1. Initialization
2. Early exaggeration
3. Normal gradient descent.

In the initialization phase, embedding points are placed according to one of various methods, which will be discussed in Section 2.2.2. In the next phase, early exaggeration, gradient descent is performed but the attractive force between points is artificially increased to aid cluster separation (see Section 2.2.3). The last phase consists of gradient descent with the original levels of repulsive force, refining the shape of the clusters (see Section 2.2.2).

2.2.1 t-SNE Cost Function

The cost function for t-SNE encourages the high-dimensional neighbors of a data-point in the dataset to correspond to the low-dimensional neighbors of the embedded point.

The *similarity* $p_{j|i}$ of datapoints x_i and x_j is given by

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

and represents the conditional probability that point x_i would choose point x_j as a neighbor, if neighbors were selected proportional to the density of a Gaussian with mean x_i and standard deviation σ_i [31]. The probabilistic interpretation can

be difficult to understand, so it may be easier to view the Gaussian density as a similarity kernel which turns pairwise distances into a value $p_{j|i} \in [0, 1]$, where values close to zero imply dissimilarity and values close to one are highly similar.

t-SNE roughly tries to mimic methods which attempt to preserve the k -nearest neighbors graph. An effective neighbor of x_i in t-SNE can be thought of as a datapoint x_j where $p_{j|i}$ is a non-negligible value. The analog of k from k -nearest neighbors is the *perplexity* hyperparameter. Perplexity is defined as

$$\text{Perp}(P_i) = 2^{H(P_i)}$$

where $H(P_i)$ represents the Shannon entropy of P_i measured in bits [31]:

$$H(P_i) = - \sum p_{j|i} \log_2(p_{j|i}).$$

The perplexity gives a continuous analog to the value k in k -nearest neighbors. In fact, in the case where a datapoint has k effective neighbors that are equidistant, then $p_{j|i}$ is a uniform distribution over these neighbors and the perplexity is exactly k :

$$\text{Perp}(P_i) = 2^{H(P_i)} = 2^{-k(1/k) \log_2(1/k)} = k$$

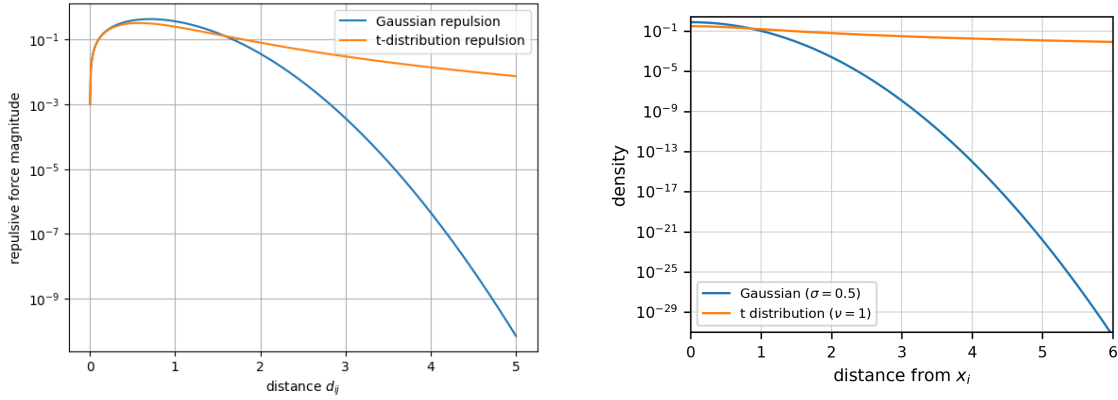
One issue with preserving the k -nearest neighbor graph is that datapoints in dense regions may have many more than k effective neighbors while datapoints in sparse regions may have far fewer than k effective neighbors. In order to adjust for the neighborhood density, the parameters σ_i are assigned to each datapoint to control the number of effective neighbors. For x_i in dense regions a small σ_i is chosen and for sparse regions a large σ_i is chosen. For each datapoint, t-SNE selects a σ_i through binary search, finding a value of σ_i where $2^{H(P_i)} = p$. Here p is a fixed perplexity hyperparameter (usually between 5 and 50) [31].

Similarly to the similarity between the high-dimensional datapoints, the embedding points also have a measure of similarity. Specifically, the similarities q_{ij} between embedding points y_i and y_j are given by

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

Note that the formulation for q_{ij} is similar to p_{ij} , but with two notable differences: q_{ij} is symmetric by construction and the Student's t-distribution density is used instead of the Gaussian density. The use of the t-distribution is an improvement over the previous algorithm SNE which used a Gaussian density for both the datapoints and the embedding.

The t-distribution is used as it alleviates the crowding problem [31]. The crowding problem is that points crowd together and collapse into highly discrete and non-smooth clusters. The crowding problem is a result of the geometry of high-dimensional space and the choice of the Gaussian density as a low-dimensional distribution. In high-dimensional space the distribution of distances from a datapoint to its neighbors is highly concentrated around a moderate distance. The Gaussian, however, assigns negligible probability to moderate distances leading to a lack of repulsive forces between the embedding points, causing crowding. The solution proposed by t-SNE is to use a t-distribution which has fatter tails than Gaussian,



(a) Magnitudes of repulsive forces of SNE and t-SNE.

(b) Tails of Gaussian ($\sigma = 0.5$) and t-distribution ($\nu = 1$).

Figure 2.3: (a) Using a Gaussian for the low-dimensional probability distribution (SNE) leads to moderate distances d_{ij} contributing negligible force, in distinction to the t-distribution (t-SNE). The repulsive force using a Gaussian is $\propto d_{ij}e^{-d_{ij}^2}$ whereas the repulsive force using a t-distribution is $\propto d_{ij}(1 + d_{ij}^2)^{-2}$. (b) The t-distribution has fatter tails than the Gaussian.

allowing points at a moderate distance to have non-negligible repulsive forces (see Figure 2.3).

Note that the high-dimensional similarity is not symmetric, i.e. $p_{j|i} \neq p_{i|j}$ in general due to the normalization factor and also the fact that in general $\sigma_i \neq \sigma_j$. t-SNE uses *symmetrized similarities* which serve to make the final cost function gradient simpler, and also experimentally produce slightly improved visualizations [31]. Symmetrized similarities p_{ij} are calculated as

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n},$$

where n is the number of datapoints. This normalization factor $\frac{1}{n}$ ensures that $\sum_{i \neq j} p_{ij} = 1$.

Both p_{ij} and q_{ij} define joint probability distributions between pairs, and their discrepancy is used as a cost function $\mathcal{C}$. Specifically, the discrepancy is measured using Kullback-Leibler (KL) divergence:

$$\mathcal{C} = KL(P \parallel Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

The KL Divergence strongly penalizes embedding points missing true neighbors, while weakly penalizing having false neighbors. In the case of missing true neighbors, p_{ij} will be large and q_{ij} will be small. Therefore the term $p_{ij} \log(\frac{p_{ij}}{q_{ij}})$ will also be large. In the case of having false neighbors, p_{ij} is small but q_{ij} is large. Therefore $p_{ij} \log(\frac{p_{ij}}{q_{ij}})$ will be small. The KL divergence is a member of a family of so-called f -divergences, which measure the difference of one probability distribution from the other. The KL Divergence, in particular, has been found to excel in revealing the cluster structure of data [21].

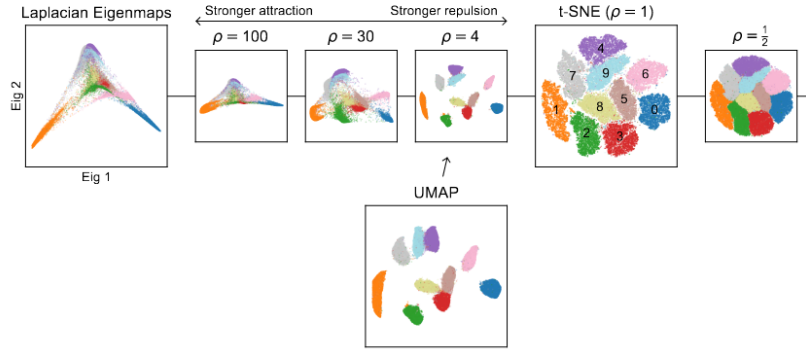


Figure 2.4: Attraction-repulsion spectrum of t-SNE for varying exaggeration factors ρ on the MNIST dataset [5].

2.2.2 Initialization and Gradient Descent

In the initialization phase, for each datapoint a corresponding point is chosen in the low-dimensional embedding space. The initialization method mentioned in the original paper was to generate random points in a small radius around the origin by sampling from a zero-mean isotropic Gaussian with small variance [31]. Later research has found that using PCA embeddings as the initial embedding points is superior to random initialization, since some semblance of the global structure is preserved [23]. When using PCA for initialization, the PCA features are often scaled to have a standard deviation of 10^{-4} [23].

After initialization, the embeddings are optimized using gradient descent. The gradient is given by the expression (see Appendix A),

$$\nabla_{y_i} \mathcal{C} = 4 \sum_j (p_{ij} - q_{ij})(1 + d_{ij}^2)^{-1}(y_i - y_j),$$

where $d_{ij} = \|y_i - y_j\|$. Then for the gradient descent the original authors proposed to add an additional momentum term to speed up convergence. The gradient update is given as follows:

$$y_i^{(t+1)} = y_i^{(t)} - \eta \nabla_{y_i} \mathcal{C} + \alpha(t)(y_i^{(t)} - y_i^{(t-1)}),$$

where η is the learning rate and $\alpha(t)$ is the momentum factor. For implementation details, we refer to scikit-learn, a widely used open-source machine learning library for Python [35]. The momentum factor $\alpha(t)$ slightly varies between that of the original paper and the scikit-learn implementations [32] [35]. In scikit-learn the momentum term is implemented as being a low-value for the early exaggeration phase and a higher value for the normal gradient optimization phase:

$$\alpha(t) = \begin{cases} 0.5, & t < 250 \quad (\text{early exaggeration}), \\ 0.8, & t \geq 250 \quad (\text{normal gradient steps}). \end{cases}$$

2.2.3 Early Exaggeration

Ignoring momentum, the embedding points move in the direction of the negative gradient. This expression for the negative gradient can be divided into two terms, one which attracts point y_i to y_j and the other which repels point y_i from y_j :

$$-\nabla_{y_i} \mathcal{C} = 4\rho \sum_j \underbrace{-p_{ij}(1 + d_{ij}^2)^{-1}(y_i - y_j)}_{\text{attractive force}} + 4 \sum_j \underbrace{q_{ij}(1 + d_{ij}^2)^{-1}(y_i - y_j)}_{\text{repulsive force}}. \quad (2.1)$$

In early exaggeration (for scikit-learn this is the first 250 iterations) the attractive force is multiplied by an exaggeration factor $\rho > 1$, resulting in widely separated tight clusters [31] [35]. The choice of ρ determines the character of the embedding: increased attraction (i.e, large ρ values) leads to discrete and clearly separated clusters, whereas decreased attraction (i.e, small values of ρ) leads to continuous clusters with fuzzier borders [5]. Other non-linear dimensionality algorithms can be viewed as fitting on an attraction-repulsion spectrum, where UMAP roughly behaves as t-SNE with $\rho = 4$ and Laplacian Eigenmaps resembles the limiting case of t-SNE where $\rho \rightarrow \infty$ (see Figure 2.4) [5].

2.3 Related work on t-SNE

In this section we will discuss improvements of and variations on the t-SNE algorithm.

2.3.1 Barnes-Hut acceleration

Barnes-Hut t-SNE is an acceleration scheme for t-SNE reducing the computational complexity of t-SNE from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$ [29]. As shown in Equation (2.1), the t-SNE gradient decomposes into an attractive and a repulsive term:

$$\nabla_{y_i} \mathcal{C} = 4(F_{\text{attr},i} - F_{\text{rep},i}).$$

The Barnes-Hut t-SNE algorithm accelerates each term separately, using a different approximation strategy for each part.

Attractive forces. The attractive force is given by

$$F_{\text{attr},i} = \sum_j p_{ij}(1 + d_{ij}^2)^{-1}(y_i - y_j).$$

Since p_{ij} depends only on the (static) high-dimensional datapoints rather than on the (moving) embedding, it needs to be computed only once and can be cached for the remainder of the optimization. Moreover, because p_{ij} decays exponentially with distance, only the nearest neighbors j of a point i contribute meaningfully, so the sum over all N points can be truncated to a sum over the k nearest neighbors:

$$\sum_j p_{ij}(1 + d_{ij}^2)^{-1}(y_i - y_j) \approx \sum_{j \in \mathcal{N}_i} p_{ij}(1 + d_{ij}^2)^{-1}(y_i - y_j).$$

Empirically, $k = \lfloor 3p \rfloor$ is a good choice, where p is the perplexity hyperparameter [29]. This k -nearest-neighbor search is performed efficiently via a depth-first search on a vantage-point tree [29]. Since p_{ij} has only $\mathcal{O}(pN)$ nonzero entries once precomputed, evaluating the attractive force at each iteration costs $\mathcal{O}(pN)$. This leaves the repulsive force as the remaining bottleneck.

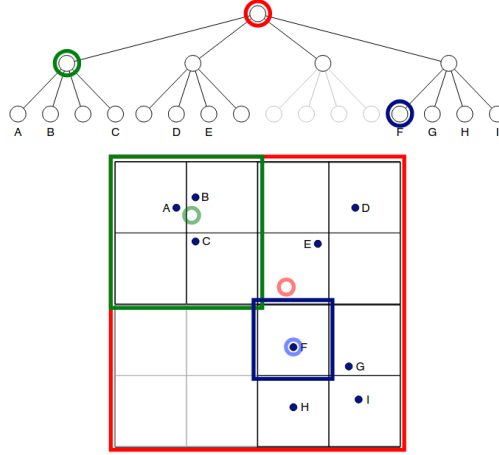


Figure 2.5: Illustration of a quadtree from [29]. Here the dataset consists of nine points, labeled by letters. The colored nodes in the tree correspond to the outlined square cells. The centers of mass of the cells are drawn as unfilled circles.

Repulsive forces. The repulsive force is given by,

$$F_{\text{repl},i} = \sum_j q_{ij} (1 + d_{ij}^2)^{-1} (y_i - y_j),$$

and can alternatively be written as

$$\sum_j q_{ij}^2 Z (y_i - y_j),$$

where Z is the normalization constant given by

$$Z = \sum_{i \neq j} q_{ij}.$$

The key idea behind accelerating the repulsive force is that distant points make near-identical contributions to one another. Concretely, if y_i is far from a cluster of points $y_j, \dots, y_k$, and those points are close to one another, then $\|y_i - y_j\| \approx \|y_i - y_k\|$, so each point in the cluster contributes almost the same repulsive force to y_i [29]. Rather than summing these near-identical contributions individually, the whole cluster can be summarized by its center of mass y_{cell} and its point count N_{cell} , so that for all j in the cluster,

$$\sum_j q_{ij}^2 Z (y_i - y_j) \approx N_{\text{cell}} q_{i,\text{cell}}^2 Z (y_i - y_{\text{cell}}),$$

with Z approximated in the same way.

The Barnes-Hut algorithm identifies these clusters by building a quadtree over the current embedding. Each non-leaf node is a rectangular cell split into four quadrants; each leaf node contains zero or one embedding point. Every node stores the center of mass and point count of the embedding points beneath it. The quadtree has $\mathcal{O}(N)$ nodes and can be constructed in $\mathcal{O}(N)$ time [29].

Whether a given cell is summarized as a single unit for a target point i , rather than descended into further, is decided by the Barnes-Hut stopping criterion:

$$\frac{r_{\text{cell}}}{\|y_i - y_{\text{cell}}\|} < \theta,$$

where r_{cell} is the diagonal length of the cell (zero for individual datapoints), and θ is a parameter controlling the coarseness of the approximation [3] [29]. Intuitively, a cell can be summarized once it is small relative to its distance from y_i : at $\theta = 0$, only zero-width cells (i.e., individual datapoints) satisfy the criterion, recovering the exact, unapproximated method [29].

Complexity. For a well-distributed set of points, each target point interacts with roughly $\mathcal{O}(\log N)$ cells, so the repulsive force computation costs $\mathcal{O}(N \log N)$ overall. This approximation introduces only a small error in practice; increasing θ trades a coarser approximation (larger error) for a faster runtime.

2.3.2 Other related methods

Here we will discuss methods which are superficially related to our goal of t-SNE acceleration, but for reasons which will be explained are not relevant.

Hierarchical-SNE (HSNE) computes t-SNE on a subset of the data, called *landmarks*, and interpolates the positions of the remaining points relative to these landmarks [36]. Despite the name, the “hierarchy” in HSNE refers to a multi-scale strategy for *visualizing* the data rather than any claim that the underlying data itself has hierarchical structure, which is the assumption motivating hyperbolic t-SNE in this thesis. A related, non-hierarchical landmark technique, Landmark-tSNE, was already proposed by van der Maaten and Hinton [31], which selects landmarks and estimates similarities between them via random walks on the k-nearest-neighbor graph.

A-tSNE accelerates the computation of the *attractive* forces through approximation, using a Forest of Randomized KD-Trees to compute fast, approximate k-nearest-neighbors for constructing the similarity probabilities $p_{j|i}$ [37]. This is useful during the initialization stage but is not directly relevant to this work, which instead accelerates the *repulsive* force computation.

GPGPU-tSNE is a GPU-based t-SNE approximation that achieves $\mathcal{O}(N)$ complexity by estimating the repulsive forces between datapoints through kernel-texture splatting [38]. Rather than computing the force between each pair of points separately, GPGPU-tSNE constructs a vector field $\mathcal{V}$, representing the repulsive force, and a scalar field $\mathcal{S}$, used to compute the normalizing constant Z . Each datapoint contributes to these fields by drawing a precomputed kernel texture centered on its position, through GPU textured-rectangle rendering with additive accumulation [38]. The resulting field values are then interpolated back to the datapoints also via the GPU rendering pipeline. While asymptotically efficient, this approach is optimized specifically for GPU hardware rather than the CPU, making it a poor fit for a CPU-based implementation. Additionally, the rendering pipeline is optimized for Euclidean operations, and may not be amenable to hyperbolic space.

FMM-tSNE accelerates t-SNE using the Fast Multipole Method [14]. Van der Maaten originally considered applying FMM to t-SNE for its favorable accuracy and

error bounds, but noted that it was not known whether the Student-t kernel used by t-SNE admits an expansion suitable for FMM [29]. Later it was shown that such an expansion does in fact exist [14], but this result was published only after we had already committed to a FIt-SNE-based acceleration scheme, and so this thesis does not build on it. Furthermore the method of [14] is not yet established, and as such it may be premature to extend it to hyperbolic space.

2.3.3 FIt-SNE

We chose to base our t-SNE acceleration off of the widely used Fast Fourier Transform-accelerated Interpolation-based t-SNE (FIt-SNE), and in this section we will describe the algorithm in detail [27]. FIt-SNE accelerates t-SNE by combining polynomial interpolation with the Fast Fourier Transform, reducing the per-iteration cost of computing embeddings to $\mathcal{O}(N)$.

The same attractive/repulsive split as Barnes-Hut t-SNE is used in FIt-SNE (Section 2.3.1):

$$\nabla_{y_i} \mathcal{C} = 4(F_{\text{attr},i} - F_{\text{rep},i}).$$

The attractive force is computed sparsely via nearest-neighbor approximation and needs to be computed only once, exactly as described in Section 2.3.1. The remainder of this section therefore focuses entirely on FIt-SNE's contribution: reducing the computation of the repulsive force $F_{\text{rep},i}$ from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$.

Kernel sums The key step is to express $F_{\text{rep},i}$ purely in terms of convolution-like *kernel sums* of the form

$$\phi_{K,f}(y_i) = \sum_{j=1}^N K(y_i, y_j) f_j,$$

where the kernel K is either

$$K_1(y_i, y_j) = \frac{1 - \delta_{ij}}{1 + \|y_i - y_j\|^2}, \quad K_2(y_i, y_j) = \frac{1 - \delta_{ij}}{(1 + \|y_i - y_j\|^2)^2},$$

with δ_{ij} the Kronecker delta, and the source f is one of

$$f_{1,j} = 1, \quad f_{2,j} = y_j(1), \quad f_{3,j} = y_j(2),$$

where $y_j(k)$ denotes the k -th component of y_j .

K_1 is used to compute the normalizing constant Z :

$$Z = \sum_{i \neq j} (1 + \|y_i - y_j\|^2)^{-1} = \sum_{i,j} K_1(y_i, y_j) = \sum_i \sum_j K_1(y_i, y_j) f_{1,j} = \sum_i \phi_{K_1, f_1}(y_i).$$

K_2 supplies the directional component of the repulsive force:

$$\begin{aligned} F_{\text{rep},i}(1) &= \frac{1}{Z} \left(y_i(1) \phi_{K_2, f_1}(y_i) - \phi_{K_2, f_2}(y_i) \right), \\ F_{\text{rep},i}(2) &= \frac{1}{Z} \left(y_i(2) \phi_{K_2, f_1}(y_i) - \phi_{K_2, f_3}(y_i) \right). \end{aligned}$$

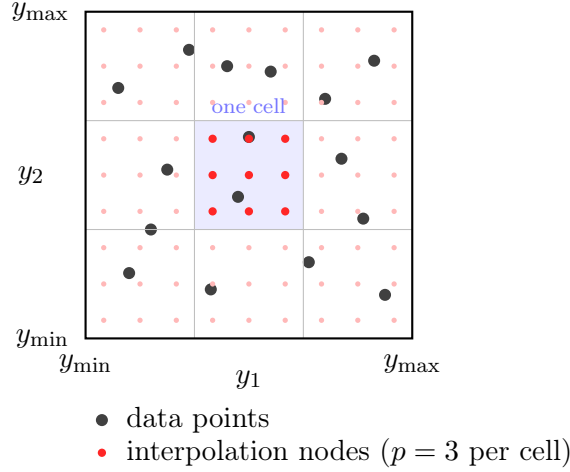


Figure 2.6: The interpolation grid used by FIt-SNE. The bounding box of the embedding is subdivided into equal-sized tiles, each populated with $p = 3$ equispaced interpolation nodes per dimension.

Computing these kernel sums directly costs $\mathcal{O}(N^2)$, which by itself accelerates nothing. FIt-SNE instead reduces this cost in two stages: first, *polynomial interpolation* onto a grid reduces the cost to be linear in N ; second, the resulting grid-based sum is evaluated using the *Toeplitz structure* of the kernel matrix via the FFT, allowing higher accuracy at no extra asymptotic cost. The remainder of this subsection covers the interpolation step; the FFT step is developed separately in Section 2.3.4.

Interpolation pipeline Rather than evaluating $\phi_{K,f}$ directly between the N scattered embedding points, FIt-SNE evaluates it between interpolation nodes on a regular grid, where the convolution structure exploited in Section 2.3.4 becomes available. The grid is constructed by covering the bounding box of the current embedding with equal-sized square tiles, each endowed with p equispaced interpolation nodes per dimension (Figure 2.6).

Interpolation then proceeds in three steps:

1. **Spread:** the source values f_j are interpolated from the datapoints onto the interpolation nodes, $\mathbf{w} = \mathbf{B}^T \mathbf{f}$, where $\mathbf{B}$ is the interpolation matrix and $\mathbf{w}$ the resulting values at the nodes.
2. **Sum:** the kernel sum is evaluated only between interpolation nodes, $\mathbf{u} = \mathbf{K} \mathbf{w}$.
3. **Gather:** the node values $\mathbf{u}$ are interpolated back to the original embedding points, giving the approximate kernel sum $\hat{\phi} \approx \phi$.

Polynomial interpolation Interpolation within each tile is performed using Lagrange polynomials. Rather than fitting a single high-degree polynomial across the whole grid, FIt-SNE uses a low-degree *piecewise* polynomial, fit independently per tile. A single high-degree interpolant on equispaced nodes is prone to the Runge phenomenon, large oscillations near the edges of the interpolation domain, which worsen as more nodes are added. Using a piecewise polynomial with a small, fixed number of nodes per tile ($p = 3$ is typical) avoids this. Here p denotes the number of interpolation points per dimension, not the degree of the interpolating polynomial.

Given a bound ε on the pointwise polynomial interpolation error within a tile, an equivalent bound can be shown for the resulting kernel-sum approximation (see Appendix B.2.1):

$$|\tilde{\phi}(y_i) - \phi(y_i)| \leq \varepsilon \sum_{j=1}^N |f_j|.$$

Computational complexity Interpolating to and from the grid nodes (steps 1 and 3 above) costs $\mathcal{O}(Np^2)$. Directly evaluating the kernel sum between grid nodes (step 2) costs $\mathcal{O}((Tp^2)^2)$, where T is the number of grid tiles. Combining these gives an overall cost of

$$\mathcal{O}(Np^2 + (Tp^2)^2)$$

using interpolation alone. As shown in Section 2.3.4, exploiting the Toeplitz structure of the kernel matrix via the FFT reduces the node-to-node summation step to $\mathcal{O}(Tp^2 \log(Tp^2))$, giving

$$\mathcal{O}(Np^2 + (Tp^2) \log(Tp^2)).$$

Both expressions are linear in N ; the benefit of the FFT-accelerated version is that, for a fixed computational budget, it permits a finer grid (larger T , smaller tiles) and therefore higher accuracy.

2.3.4 Use of the Fast Fourier Transform in FIt-SNE

The node-to-node kernel sum $\mathbf{u} = \mathbf{K}\mathbf{w}$ introduced above currently costs $\mathcal{O}((Tp^2)^2)$ to evaluate directly. We now show how two structural properties allow this to be reduced to $\mathcal{O}(Tp^2 \log(Tp^2))$ via the FFT. We derive this for a 1D embedding, where the argument is cleanest.

Two properties of the kernel sum First, *translation invariance*: both K_1 and K_2 depend only on $\|y_i - y_j\|$, so

$$K(y_i + v, y_j + v) = K(y_i, y_j) \quad \text{for any shift } v,$$

and in particular $K(y_i, y_j) = K(y_i - y_j, 0)$. This lets us write the kernel as a function of a single difference, $\kappa(y_i - y_j) := K(y_i - y_j, 0)$.

Second, *a uniform grid*: the interpolation nodes constructed above are equispaced, $y_i = y_0 + i \Delta y$, so that $\kappa(y_i - y_j) = \kappa((i - j)\Delta y) =: \kappa_{i-j}$.

Together, these let us rewrite the node-to-node kernel sum as

$$\phi(y_i) = \sum_j \kappa_{i-j} f_j,$$

a *discrete convolution* of $\mathbf{f}$ with $\boldsymbol{\kappa}$.

Convolution using the FFT To see exactly why this convolution can be evaluated, it helps to write it as a matrix-vector product. Let $\mathbf{M}$ be the matrix with entries $M_{ij} = \kappa_{i-j}$, so that $\phi = \mathbf{M}\mathbf{f}$. Since M_{ij} depends only on the difference $i - j$, every diagonal of $\mathbf{M}$ is constant: $\mathbf{M}$ is precisely a *Toeplitz matrix*. In other words, translation invariance and the uniform grid are exactly the two conditions that make the node-to-node kernel matrix Toeplitz.

A matrix $\mathbf{T} \in \mathbb{R}^{n \times n}$ is Toeplitz if $T_{ij} = t_{i-j}$ for some sequence $(t_k)_{k=-(n-1)}^{n-1}$ [18]:

$$\mathbf{T} = \begin{pmatrix} t_0 & t_{-1} & \cdots & t_{-(n-1)} \\ t_1 & t_0 & \cdots & t_{-(n-2)} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n-1} & t_{n-2} & \cdots & t_0 \end{pmatrix}.$$

Toeplitz structure alone does not reduce the cost of computing $\mathbf{M}\mathbf{f}$, a direct matrix-vector product still costs $\mathcal{O}(n^2)$. The remaining step is to embed $\mathbf{T}$ into a *circulant* matrix that the FFT can act on directly.

A circulant matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$ is a Toeplitz matrix in which each row is a cyclic shift of the previous one, $C_{ij} = c_{(i-j) \bmod n}$, and is therefore fully determined by its first column alone.

Any Toeplitz matrix $\mathbf{T}$ can be embedded into a circulant matrix $\mathbf{C}$ [18]. The reason this embedding is useful is that circulant matrices are diagonalized by the discrete Fourier transform:

$$\mathbf{C} = \mathbf{F}_n^{-1} \text{diag}(\mathbf{F}_n \mathbf{c}) \mathbf{F}_n$$

[18], where $\mathbf{F}_n$ is the $n \times n$ DFT matrix. Computing $\mathbf{C}\mathbf{x}$ therefore reduces to

$$\tilde{\mathbf{x}} = \mathbf{F}_n \mathbf{x}, \quad \tilde{\mathbf{c}} = \mathbf{F}_n \mathbf{c}, \quad \mathbf{z} = \tilde{\mathbf{c}} * \tilde{\mathbf{x}}, \quad \mathbf{y} = \mathbf{F}_n^{-1} \mathbf{z},$$

i.e., two forward FFTs (one of which, $\mathbf{F}_n \mathbf{c}$, can be precomputed once per iteration and reused for every point), one inverse FFT, and an elementwise product. Since $\mathbf{F}_n \mathbf{x}$ is computed via the Fast Fourier Transform algorithm in $\mathcal{O}(n \log n)$ rather than $\mathcal{O}(n^2)$, the entire node-to-node convolution costs $\mathcal{O}(n \log n)$.

Combining this with the interpolation cost from the previous subsection, the full repulsive-force computation costs

$$\mathcal{O}(Np^2 + (Tp^2) \log(Tp^2)),$$

as stated above: the interpolation step is linear in N , and the FFT-accelerated grid convolution is independent of N for a fixed grid resolution, so the overall cost remains $\mathcal{O}(N)$.

2.4 Hyperbolic Space

Now that we have discussed the concepts related to t-SNE in Euclidean space, we will explain hyperbolic space to make hyperbolic t-SNE understandable.

To motivate hyperbolic space we will note that recalling Euclidean space is poor at embedding exponentially growing data structures like trees, as their number of nodes increase exponentially, but Euclidean space volume increases polynomially. In contrast, hyperbolic spaces have the property that volume increases exponentially with radius [34]. More specifically, the volume of a ball of radius r in n -dimensional Euclidean space grows polynomially,

$$V_n^{\mathbb{E}}(r) \propto r^n,$$

whereas in n -dimensional hyperbolic space $\mathbb{H}^n$ it grows exponentially [40],

$$V_n^{\mathbb{H}}(r) \propto e^{(n-1)r}.$$

This exponential growth property allows tree-like structures to be embedded accurately in low-dimensional hyperbolic spaces [41]. In particular, even two dimensional hyperbolic space can accurately represent large hierarchies, making it well suited for visualization tasks.

2.4.1 Overview of Hyperbolic Space

Hyperbolic space in n dimensions $\mathbb{H}^n$ can be defined through the *hyperboloid model*, where points lie on a manifold embedded in $\mathbb{R}^{n+1}$. In this model, $\mathbb{R}^{n+1}$ is equipped with the *Lorentzian bilinear form* between two vectors $\Theta, \Phi \in \mathbb{R}^{n+1}$, given by [47]:

$$\langle \Theta, \Phi \rangle_{n:1} = -\Theta_{n+1}\Phi_{n+1} + \sum_{i=1}^n \Theta_i\Phi_i.$$

The points in this model lie on the manifold given by [47],

$$\mathbb{H}^n = \{ \Theta \in \mathbb{R}^{n+1} \mid \langle \Theta, \Theta \rangle_{n:1} = -1, \Theta_{n+1} > 0 \}.$$

This model defines a complete, simply connected Riemannian manifold with a constant negative curvature [8]. While the Lorentzian bilinear form $\langle \cdot, \cdot \rangle_{n:1}$ is indefinite on $\mathbb{R}^{n+1}$, its restriction to each tangent space $T_{\Theta}\mathbb{H}^n$ is positive definite, and therefore defines a genuine inner product [47]. This restriction is precisely the Riemannian metric on $\mathbb{H}^n$. One can show from the metric that the geodesic distance between two points $\Theta, \Phi \in \mathbb{H}^n$ in the hyperboloid model is given by [47],

$$d_{\mathbb{H}^n}(\Theta, \Phi) = \operatorname{arccosh}(-\langle \Theta, \Phi \rangle_{n:1}).$$

The hyperboloid model is well suited for analytic calculations [47]. However, it is not geometrically intuitive. Distances are computed using the Lorentzian inner product, so the intrinsic geometry of the surface does not correspond to its Euclidean appearance. Luckily, there are many models of hyperbolic space equivalent to the hyperboloid model, which allow better visual intuition. In particular we will focus on the Poincaré disk model of $\mathbb{H}^2$ for visualization.

In the *Poincaré disk model*, points are embedded on the unit disk,

$$\mathbb{D} = \{y \in \mathbb{R}^2 \mid y_1^2 + y_2^2 < 1\},$$

and $\mathbb{D}$ is equipped with the following metric:

$$g_y^{\mathbb{D}} = \frac{4}{(1 - \|y\|^2)^2} g^E,$$

where g^E is the standard Euclidean metric [44]. Note here that the metric $g_y^{\mathbb{D}}$ varies with the distance from the center of the disk, implying distances and areas near the edge of the disk are larger than they appear (see Figure 2.7).

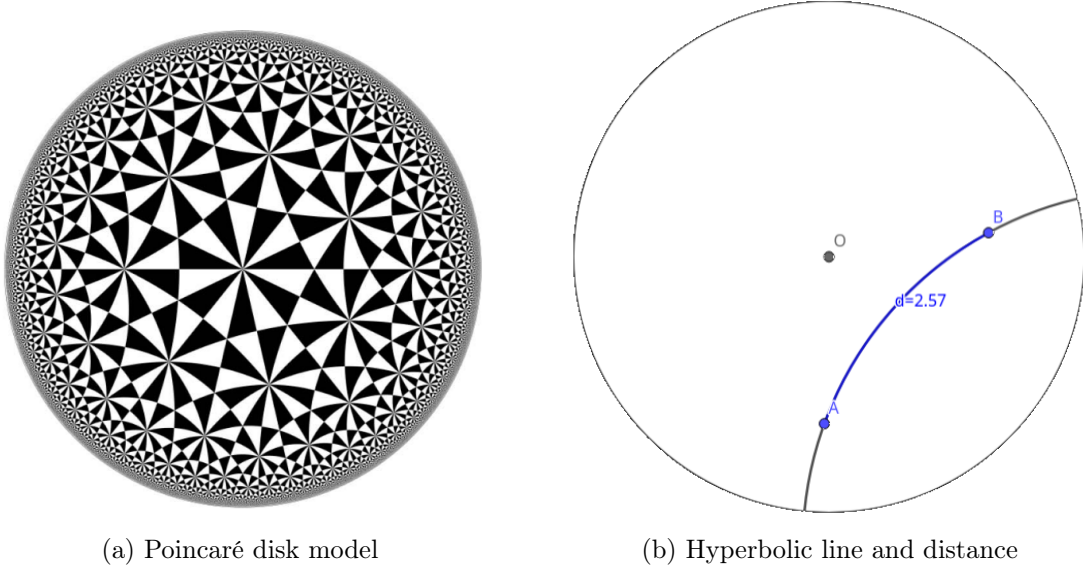


Figure 2.7: (a) Tiling of the Poincaré disk by equal sized triangles [44]. Note that although the triangles appear to be smaller near the edge of the disk, in fact all triangles have the same area under the associated metric of the Poincaré disk. (b) Geodesic between points A and B on the Poincaré disk (blue) and the hyperbolic line passing through them (gray) [17]. The inward curvature of the geodesic reflects the non-Euclidean metric. The hyperbolic distance d exceeds the corresponding Euclidean distance.

Geodesics between two points take the form of circular arcs which curve inward towards the origin (see Figure 2.7). The hyperbolic distance $d_{ij}^{\mathcal{H}}$ between two points y_i and y_j on the Poincaré disk is given by:

$$d_{ij}^{\mathcal{H}} = \operatorname{arccosh} \left(1 + 2 \frac{\|y_i - y_j\|^2}{(1 - \|y_i\|^2)(1 - \|y_j\|^2)} \right). \quad (2.2)$$

There is a simple and explicit mapping between the hyperboloid model and the Poincaré disk model (see Figure 2.8). The mapping from the Poincaré disk point y to the corresponding point Θ on the hyperboloid is given by,

$$\Theta = \frac{2}{1 - \|y\|^2} (y_1, \dots, y_n, \frac{1 + \|y\|^2}{2})$$

whereas the reverse mapping is given by,

$$y = \frac{1}{\Theta_{n+1} + 1} (\Theta_1, \dots, \Theta_n).$$

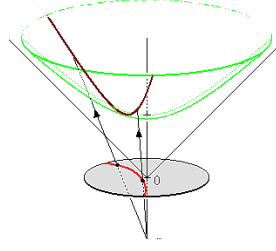


Figure 2.8: Correspondence between the hyperboloid and Poincaré models. The geodesic on the hyperboloid (brown), projects onto a circular arc (red) on the Poincaré disk [48].

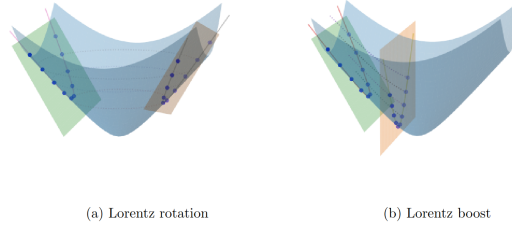


Figure 2.9: Illustration of the decomposition of the two components of a Lorentz transformation [12]. Note that the blue dots maintain the same distance between themselves under the transformation since the Lorentz transformation is an isometry.

2.4.2 Lorentz Transformations

Lorentz transformations are a group of distance-preserving transformations on $\mathbb{H}^2$, analogous to rotations in Euclidean space. We will use Lorentz transformations to construct a tiling of the hyperbolic plane (see Section 3.2.1) and for simplifying polynomial interpolation (see Section 3.2).

Definition A function $\Lambda : \mathbb{H}^n \rightarrow \mathbb{H}^n$ is a Lorentz transformation iff [40]

$$\langle \Lambda(\Theta), \Lambda(\Phi) \rangle_{n:1} = \langle \Theta, \Phi \rangle_{n:1} \quad \forall \Theta, \Phi \in \mathbb{H}^n$$

It can be shown that Λ must be linear and its matrix representation $\mathbf{A}$ is Lorentz orthogonal [40] (i.e. $\mathbf{A}^T \mathbf{J} \mathbf{A} = \mathbf{J}$ where $\mathbf{J} = \text{diag}(1, \dots, 1, -1)$).

Polar decomposition For constructing the tiling of $\mathbb{H}^2$, each tile is the image of some base tile under a Lorentz transformation; and as such each tile is the same size, and it is simple to map to the base tile and back. Since the tiling is constructed radially outward from the origin, it is natural to work with a hyperbolic analogue of polar coordinates.

For $n = 2$, any Lorentz transformation Λ with matrix representation $\mathbf{A}$ can be written [15] as

$$\mathbf{A} = \mathbf{R}(m) \mathbf{B}(r) \mathbf{R}(n), \quad \mathbf{R}(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{B}(r) = \begin{pmatrix} \cosh r & 0 & \sinh r \\ 0 & 1 & 0 \\ \sinh r & 0 & \cosh r \end{pmatrix},$$

where $\mathbf{R}(\theta)$ is a rotation in the (Θ_1, Θ_2) plane, fixing the timelike coordinate Θ_3 , and $\mathbf{B}(r)$ is a Lorentz boost — the hyperbolic equivalent of a translation — by distance r along the Θ_1 axis (see Figure 2.9).

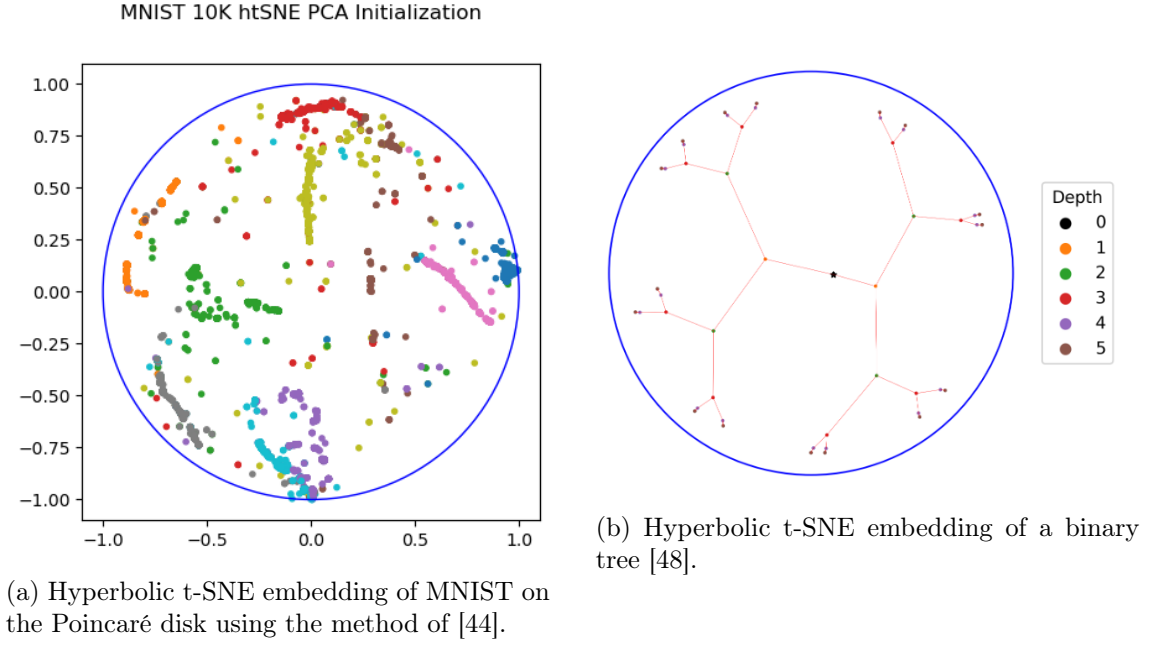


Figure 2.10: Examples of hyperbolic t-SNE embeddings.

Here we can define an orientation preserving Lorentz transformation by the trio (m, r, n) with inverse $(-n, -r, -m)$.

2.5 Hyperbolic t-SNE

The ability of hyperbolic spaces to embed hierarchical data with low distortion motivates an extension of t-SNE to hyperbolic space, which we will call *hyperbolic t-SNE* (see Figure 2.10). We will first explain the basic concepts of hyperbolic t-SNE through the method of Skrodzki et al. in Section 2.5.1 [44]. In Section 2.5.2, we will then note the variations in approaches to hyperbolic t-SNE.

2.5.1 Basic method of hyperbolic t-SNE

In the following section, we will describe the approach of Skrodzki et al. [44]. This method embeds points on the Poincaré disk, and additionally all calculations are done directly in the Poincaré disk model. In this approach, the high-dimensional symmetrized similarities p_{ij} remain unchanged and only the embedding similarities q_{ij} are replaced by their hyperbolic analogs $q_{ij}^{\mathcal{H}}$. Here $q_{ij}^{\mathcal{H}}$ is defined identically to q_{ij} , however, using hyperbolic distance instead of Euclidean distance:

$$q_{ij}^{\mathcal{H}} = \frac{\left(1 + (d_{ij}^{\mathcal{H}})^2\right)^{-1}}{\sum_{k \neq \ell} \left(1 + (d_{k\ell}^{\mathcal{H}})^2\right)^{-1}},$$

where $d_{ij}^{\mathcal{H}}$ is the hyperbolic distance between Poincaré embedding points y_i and y_j (see Equation (2.2)). The cost function for hyperbolic t-SNE in turn uses these hyperbolic distance similarities instead:

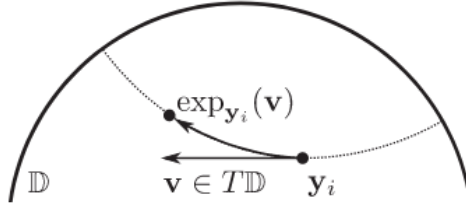


Figure 2.11: Illustration of the exponential map projecting a tangent vector v , onto the geodesic on the Poincaré disk $\mathbb{D}$ [44].

$$\mathcal{C} = KL(P \| Q^{\mathcal{H}}) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}^{\mathcal{H}}}.$$

The gradient for the hyperbolic t-SNE cost function on the Poincaré disk is given as,

$$\text{grad}_{y_i} C = 4 \sum_j (p_{ij} - q_{ij}) (1 + (d_{ij}^{\mathcal{H}})^2)^{-1} d_{ij}^{\mathcal{H}} \frac{1}{\beta_j \sqrt{\gamma^2 - 1}} [(\|y_i - y_j\|^2 + \beta_i) y_i - \beta_i y_j], \quad (2.3)$$

where $\beta_i = 1 - \|y_i\|^2$ is the distance of y_i from the edge of the disk, d_{ij} is the geodesic distance between points y_i and y_j , and $\gamma = 1 + 2 \frac{\|y_i - y_j\|^2}{\beta_i \beta_j}$ (see Appendix A for a full derivation). However, due to an algebraic error in the derivation, this correct gradient is not used in [44] or [20]. The issue is that the incorrect gradient is missing a factor of $d_{ij}^{\mathcal{H}}$, giving the incorrect gradient $\tilde{G}_{y_i}$ as,

$$\tilde{G}_{y_i} = 4 \sum_j (p_{ij} - q_{ij}) (1 + (d_{ij}^{\mathcal{H}})^2)^{-1} \frac{1}{\beta_j \sqrt{\gamma^2 - 1}} [(\|y_i - y_j\|^2 + \beta_i) y_i - \beta_i y_j].$$

The incorrect gradient $\tilde{G}$ is used for Riemannian gradient descent. Riemannian gradient descent is a generalization of gradient descent to Riemannian manifolds which accounts for the fact that for a general manifold, geodesics will not be straight lines. Whereas in regular gradient descent one adds a multiple of the gradient to the current point to get the next iterant, in Riemannian gradient descent the next iterant is calculated by using the exponential map of a multiple of the gradient.

The exponential map is a function which projects vectors from the tangent space $T_y \mathcal{M}$ of a manifold $\mathcal{M}$ to the manifold $\mathcal{M}$ itself, moving the point along the geodesic determined by the tangent vector (see Figure 2.11). The exponential map on the Poincaré disk for a tangent vector v at a point on the disk y_i is given by,

$$\exp_{y_i}(v) = y_i \oplus \left(\tanh \left(\frac{\|v\|}{1 - \|y_i\|^2} \right) \frac{v}{\|v\|} \right),$$

can be written in terms of the Möbius addition operator $\oplus$, which is defined as [44],

$$y_i \oplus y_j = \frac{(1 + 2\langle y_i, y_j \rangle + \|y_j\|^2) y_i + (1 - \|y_i\|^2) y_j}{1 + 2\langle y_i, y_j \rangle + \|y_i\|^2 \|y_j\|^2}.$$

The final form of the Riemannian gradient descent procedure can be summarized as,

$$y_i^{(t+1)} = \exp_{y_i^{(t)}}(-\eta \tilde{G}_{y_i^{(t)}}),$$

where η is the learning rate and $y_i^{(t)}$ is the current iterant. The true procedure as described in [44] also uses momentum.

2.5.2 Variations on hyperbolic t-SNE

There are multiple variations of hyperbolic t-SNE in the literature, and their salient differences will be discussed in this section.

In Klimovskaia et al., the notable innovation is the use of the symmetric Kullback-Leibler Divergence:

$$\text{KL}(P||Q^{\mathcal{H}}) + \text{KL}(Q^{\mathcal{H}}||P),$$

instead of the normal KL Divergence [22]. The symmetric divergence trades off local structure for global structure [21].

In the hyperbolic t-SNE method of Zhou and Sharpee, the cost function,

$$\text{KL}(P||Q^{\mathcal{H}}) + \lambda \text{KL}(\hat{P}||\hat{Q}),$$

is used where the second term is a regularization term that penalizes unfaithful distance values [49]. $\hat{P}$ and $\hat{Q}$ are calculated as follows [49]:

$$\hat{p}_{ij} = \frac{1 + \|x_i - x_j\|^2}{\sum_{m \neq n} (1 + \|x_m - x_n\|^2)}, \quad \hat{q}_{ij} = \frac{1 + \|y_i - y_j\|^2}{\sum_{m \neq n} (1 + \|y_m - y_n\|^2)}.$$

The parameter λ controls the balance between local and global structure preservation [49]. Large values of λ lead to clusters being positioned faithfully relative to each other, but at the cost of less distinct clustering [49].

CO-SNE (Cauchy Origin SNE) has two notable features: using a general Cauchy distribution instead of t-distribution, and adding a normalization term to preserve global hierarchy [20]. CO-SNE uses a Cauchy distribution, a family of distributions parameterized by parameter γ . When $\gamma = 1$ this is the case of the t-distribution. The justification for using a general Cauchy distribution (in the CO-SNE paper, $\gamma = 0.1$ is typically used) is that the hyperbolic t-distribution has skinny tails which causes the crowding problem, whereas using a Cauchy distribution with small γ one can obtain fatter tails [20]. This can also be seen as decreasing the attractive force between neighbors [20]. The second aspect of CO-SNE is regularization term $\mathcal{H}$ where [20],

$$\mathcal{H} = \frac{1}{m} \sum_{i=1}^m (\|x_i\|^2 - \|y_i\|^2)^2.$$

This regularization term is used in the final CO-SNE cost function:

$$\lambda_1 \text{KL}(P||Q^{\mathcal{H}}) + \lambda_2 \mathcal{H},$$

where the effects of varying hyperparameter λ_1 and λ_2 are discussed in the CO-SNE paper's supplement [20].

2.5.3 Hyperbolic Barnes-Hut acceleration

Hyperbolic Barnes-Hut acceleration [44] adapts the Euclidean Barnes-Hut acceleration described in Section 2.3.1 to hyperbolic t-SNE. Here we give a taste of what changes when moving from the Euclidean to the hyperbolic setting. Embeddings

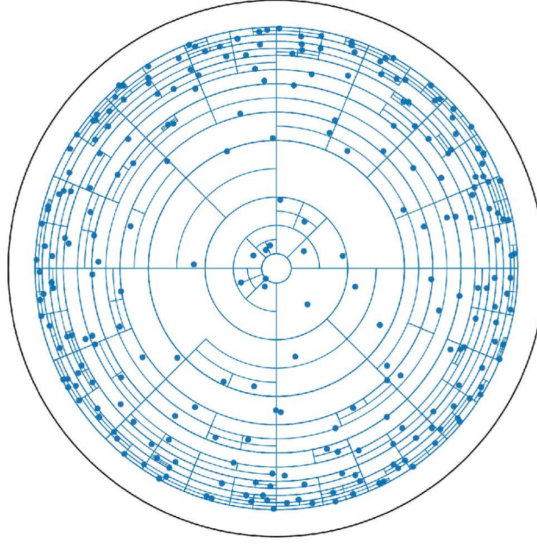


Figure 2.12: Illustration of the polar quadtree data structure [44]. The dots represent embedded points and the rectangles are the cells.

are placed in the Poincaré disk model of hyperbolic space. The Poincaré disk is used because it is conformal, meaning that angles are not distorted depending on where in the disk they are measured. This property is useful because it helps in splitting the space angularly into a hierarchy. Where Euclidean Barnes-Hut uses a quadtree, hyperbolic Barnes-Hut instead uses a *hyperbolic polar quadtree*, as developed in [28].

The polar quadtree The polar quadtree is illustrated in Figure 2.12. It is constructed as follows. The root cell of the quadtree is an annulus centered at the origin that encompasses all embedding points y_i . This root cell is then recursively subdivided into four polar quadrilaterals by splitting both the angular and radial directions. The angular split is performed at the midpoint $\text{mid}_\phi = (\max_\phi + \min_\phi)/2$, while the radial split follows the modified splitting rule $\text{mid}_r = (\max_r + \min_r)/2$, which produces equal-length radial subdivisions. This recursion continues until each leaf node contains a single embedding point.

The traversal rule for whether a contribution is to be approximated, as described in Section 2.3.1, is maintained:

$$\frac{r_{\text{cell}}}{d(y_i, y_{\text{cell}})} < \theta,$$

where d is the hyperbolic distance as defined in Equation (2.2). However, two quantities in this expression are more complicated to compute than in the Euclidean case.

The first is r_{cell} , the diameter of the cell. In the Euclidean case, this is simply the length of the cell's diagonal. In the hyperbolic case, it is instead the larger of the diagonal and the outer arc of the cell. This is because the outer arc can be larger than the diagonal when the cell is far from the origin of the disk.

The second complication is y_{cell} , the cell's center. The arithmetic mean does not exist in hyperbolic space. Instead, the natural replacement is the Fréchet mean. However, the Fréchet mean can only be computed iteratively [9]. It is therefore

more efficient to use the *pseudo-Fréchet mean* instead:

$$m(\{v_j\}) = \sum_j \left(\frac{\gamma(v_j)}{\sum_l \gamma(v_l)} \right) v_j, \quad \gamma(v_j) = \frac{1}{\sqrt{1 - \|v_j\|^2}},$$

which gives a closed-form approximation to the Fréchet mean [19].

Attractive and repulsive forces As in Euclidean Barnes-Hut, the gradient is split into attractive and repulsive forces:

$$F_{\text{attr},i} = \sum_j p_{ij} (1 + (d_{ij}^{\mathcal{H}})^2)^{-1} \frac{1}{\beta_j \sqrt{\gamma^2 - 1}} [(\|y_i - y_j\|^2 + \beta_i) y_i - \beta_i y_j],$$

$$F_{\text{repl},i} = \sum_j q_{ij} (1 + (d_{ij}^{\mathcal{H}})^2)^{-1} \frac{1}{\beta_j \sqrt{\gamma^2 - 1}} [(\|y_i - y_j\|^2 + \beta_i) y_i - \beta_i y_j],$$

where γ_{ij} and β_j are as defined in Section 2.5. The full gradient is then given by

$$\tilde{G}_{y_i} = 4(F_{\text{attr},i} - F_{\text{repl},i}).$$

The affinities p_{ij} still appear only in the attractive force. Since this is unchanged from the Euclidean case, the same sparse k -nearest-neighbor acceleration used in Euclidean Barnes-Hut (Section 2.3.1) can be applied here directly. As in the Euclidean case, this means that only the repulsive force still needs to be accelerated.

Computing the repulsive force The computation of the repulsive force proceeds in three steps.

1. The polar quadtree is constructed.
2. Z is computed by descending depth-first through the polar quadtree, starting at the root, and accumulating the contribution $N_{\text{cell}} q_{i,\text{cell}}$ whenever the traversal condition above is met.
3. For each embedding y_i , we descend depth-first through the polar quadtree, again starting at the root, accumulating the contribution

$$N_{\text{cell}} q_{i,\text{cell}}^2 Z \frac{1}{\beta_{\text{cell}} \sqrt{\gamma^2 - 1}} [(\|y_i - y_{\text{cell}}\|^2 + \beta_i) y_i - \beta_i y_{\text{cell}}],$$

to $F_{\text{repl},i}$ whenever the traversal condition is met.

This procedure exhibits an $\mathcal{O}(N \log N)$ complexity for a reasonable choice of the parameter θ [44].

2.6 Refinement of the research questions

Now that we have given sufficient background, let us return to our central research question:

Can the interpolation-based acceleration strategy of FIt-SNE be extended to hyperbolic t-SNE?

We will decompose this into three questions:

- RQ1. How can the hyperbolic t-SNE gradient be reformulated and decomposed into components suitable for interpolation-based approximation?
- RQ2. What tiling and local-coordinate scheme enables accurate polynomial interpolation on hyperbolic space, given that, unlike a Euclidean grid, a hyperbolic tiling cannot simply be refined?
- RQ3. How does HIt-SNE perform relative to the exact method in terms of runtime, approximation accuracy, and visual fidelity of the resulting embeddings?

Chapter 3

Proposed Algorithm: Hyperbolic Interpolation-based t-SNE

In this chapter, we propose a new algorithm, hyperbolic interpolation-based t-SNE (Hit-SNE), which accelerates hyperbolic t-SNE through an interpolation-based approximation. Hit-SNE adapts the Euclidean t-SNE algorithm FIt-SNE [27]. In Section 3.1 we reformulate the hyperbolic gradient from [44], into a form amenable to approximation. In Section 3.2 we introduce the tiling of hyperbolic space and the local interpolation scheme built upon it. In Section 3.3 we show how interpolation is combined with exact near-field computation to efficiently approximate the gradient. We conclude, in Section 3.4, with a full outline of the algorithm, together with an asymptotic analysis of its runtime and a discussion of its parameters.

3.1 Reformulating Hyperbolic t-SNE for Approximation

This section addresses RQ1 by reformulating the hyperbolic t-SNE gradient into a form suitable for interpolation. We first, in Section 3.1.1, reformulate the gradient in the hyperboloid model, then in Section 3.1.2, we split the gradient into repulsive and attractive forces, and lastly in Section 3.1.3, we explain how the repulsive force can be computed in terms of kernel sums.

3.1.1 Hyperboloid formulation of hyperbolic t-SNE

In current implementations of hyperbolic t-SNE the gradient is calculated in the Poincaré model (see Equation (2.3)) [44] [20]. The expression for the Poincaré gradient is unruly, and gradient descent in the Poincaré model has been found to converge more slowly than in the hyperboloid model [47].

We propose a novel formulation of the hyperbolic t-SNE gradient in the hyperboloid model. Our formulation of the gradient in the hyperboloid model is given as:

$$\nabla_{\Theta_i}^{\mathbb{H}^n} C = -4 \sum_j (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1} d_{ij} \langle \Theta_i, \Theta_j \rangle_{n:1}^2 (1 - \langle \Theta_i, \Theta_j \rangle_{n:1})^{-1/2} [\langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_i + \Theta_j],$$

where d_{ij} is the hyperbolic distance between embedding points Θ_i and Θ_j .

This form of the gradient is derived in Appendix A.1.3. Furthermore, we verified that our newly derived hyperboloid gradient does indeed correspond to the Poincaré gradient in Appendix A.2.

3.1.2 Splitting the hyperbolic t-SNE gradient into two approximation problems

The hyperboloid model gradient can be written as the difference between attractive and repulsive forces:

$$\nabla_{\Theta_i}^{\mathbb{H}^n} C = -4(F_{\text{attr},i} - F_{\text{rep},i}).$$

The attractive force $F_{\text{attr},i}$ is given as,

$$F_{\text{attr},i} = \sum_j p_{ij} (1 + d_{ij}^2)^{-1} d_{ij} (\langle \Theta_i, \Theta_j \rangle_{n:1}^2 - 1)^{-1/2} [\langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_i + \Theta_j]$$

while the repulsive force $F_{\text{rep},i}$ is given as,

$$F_{\text{rep},i} = \sum_j q_{ij} (1 + d_{ij}^2)^{-1} d_{ij} (\langle \Theta_i, \Theta_j \rangle_{n:1}^2 - 1)^{-1/2} [\langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_i + \Theta_j]$$

The high-dimensional similarities p_{ij} are constant, and as such can be computed once and reused at each gradient descent iteration [27]. As p_{ij} are also highly sparse, with most entries exactly zero, the attractive force can be approximated by summing only over the k -nearest neighbors [29].

The repulsive force admits no such simplification. The embedding similarities q_{ij} are dense and decay slowly with distance, so every pair of points contributes non-negligibly to the sum; restricting the repulsive force to a k -nearest-neighbor set would discard real contributions from distant points rather than redundant computation, introducing error rather than avoiding it. The q_{ij} are also recomputed at every iteration, since they depend on the current embedding positions, so any approximation must be efficient to repeat at each step rather than reusable across iterations. We therefore turn to a different strategy: rather than truncating the sum, we approximate it directly by adapting FIt-SNE [27] to hyperbolic space.

3.1.3 Hyperbolic kernel sums

Following FIt-SNE, the repulsive force can be efficiently approximated by expressing it as a sum of kernel-weighted source terms. For a kernel K and source values f , define [27]

$$\phi_{K,f}(y_i) = \sum_{\substack{j=1 \\ j \neq i}}^N K(y_i, y_j) f_j.$$

As in FIt-SNE, two kernels are needed: K_1 is used for the normalizing constant Z , while K_2 is used for the directional part of the force. Both are derived in Appendix B.1, and are given by

$$K_1(\Theta_i, \Theta_j) = \frac{1}{1 + d_{ij}^2}, \quad K_2(\Theta_i, \Theta_j) = \frac{d_{ij}}{(1 + d_{ij}^2)^2} (\langle \Theta_i, \Theta_j \rangle_{n:1}^2 - 1)^{-1/2}.$$

Four source terms f are used: f_1 is used for the normalizing constant, while f_2, f_3 , and f_4 correspond to the three components of a point in the hyperboloid model. These are likewise derived in Appendix B.1, and are given by

$$f_{1,j} = 1, \quad f_{2,j} = \Theta_j(1), \quad f_{3,j} = \Theta_j(2), \quad f_{4,j} = \Theta_j(3).$$

Using these, define the normalizing constant $Z = \sum_i \phi_{K_1, f_1}(\Theta_i)$ and the vector

$$X_i = (\phi_{K_2, f_2}(\Theta_i), \phi_{K_2, f_3}(\Theta_i), \phi_{K_2, f_4}(\Theta_i)).$$

The repulsive force can then be constructed from Z and X_i as

$$F_{\text{rep},i} = \frac{1}{Z} (\Theta_i \langle \Theta_i, X_i \rangle_{n:1} + X_i)$$

(see derivation in Appendix B.1).

The kernel sums above can be expressed more compactly in matrix notation. Define $(\mathbf{K}_1)_{ij} = K_1(\Theta_i, \Theta_j)(1 - \delta_{ij})$ and $(\mathbf{K}_2)_{ij} = K_2(\Theta_i, \Theta_j)(1 - \delta_{ij})$, where the $(1 - \delta_{ij})$ factor enforces the $j \neq i$ exclusion from the sums above. The normalizing constant and directional term then become

$$Z = \mathbf{1}^T \mathbf{K}_1 \mathbf{1}, \quad \mathbf{X} = \mathbf{K}_2 \Theta,$$

and the repulsive force $F_{\text{rep},i}$ takes the same form as above, with X_i and Z given by these matrix expressions.

3.2 Local interpolation on hyperbolic tiles

This section addresses RQ2. In this section we describe how the kernel sums from Section 3.1.3 can be approximated locally using polynomial interpolation. Both kernels K_1 and K_2 are smooth functions of the embedding points, so, as in FIt-SNE, it is natural to approximate the kernel sums using polynomial interpolation [27].

Adapting this scheme from FIt-SNE to hyperbolic space raises three main issues. First, tilings of hyperbolic space are difficult to generate, and even basic operations such as checking whether a point lies in a given tile are nontrivial; we address tile generation in Section 3.2.1 and tile assignment in Section 3.2.2. Second, these tilings cannot be refined, since tile size is fixed by the topology of the tiling; we recover the accuracy this costs us with a separate near-field correction in Section 3.3.2. Third, polynomials are not straightforward to define on hyperbolic space itself; we resolve this by defining them in the tangent space of each tile, as described in Section 3.2.3 and Section 3.2.4.

3.2.1 Tile mesh generation

There are many different tilings of the hyperbolic plane. A *regular* hyperbolic tiling is a tiling of the hyperbolic plane in which each tile is a regular polygon, i.e., a polygon whose internal angles are all equal. Regular tilings are parameterized by a pair (p, q) , known as the Schläfli symbol, where p is the number of sides of each tile and q is the number of tiles meeting at each vertex. In HIt-SNE, we use the $(4, 5)$ tiling, also known as the order-5 square tiling (see Figure 3.2).

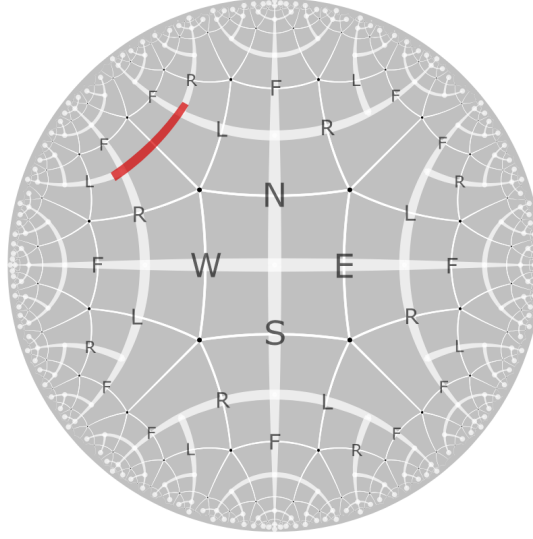


Figure 3.1: Illustration of the tiling construction [6]. The base tile is in the center. NWSE represent tile creation in the cardinal directions, and L, F, and R represent tile creation in the left, forward and right directions. The red line indicates an illegal move.

The method we use to generate the tiling is taken from [6]. In this approach, a base tile centered at the origin of the Poincaré disk is defined, and all other tiles are then constructed as images of the base tile under various Lorentzian transformations. The base tile is a square whose corners lie at distance c from the origin, where c is computed from a : half the distance between adjacent tile centers [6]. Since a tile's edge lengths are fully determined by its Schläfli symbol, both a and c can be calculated directly using hyperbolic trigonometry [6]:

$$a = \operatorname{arccosh} \left(\frac{\cos\left(\frac{\pi}{q}\right)}{\sin\left(\frac{\pi}{p}\right)} \right) \approx 0.53,$$

$$c = \operatorname{arcsinh} \left(\frac{\sinh(a)}{\sin\left(\frac{\pi}{q}\right)} \right) \approx 0.84.$$

Once the base tile is constructed, four neighboring tiles are built in the north, west, south, and east directions by applying Lorentzian transformations that rotate the base tile by a multiple of π and translate it forward by the inter-tile center distance $2a$ [6]. Starting from the base tile and these four cardinal neighbors, the remaining tiles are then constructed recursively (see Figure 3.1) [6]. At each step, a tile can generate new tiles to its left, right, or forward. To avoid generating redundant tiles, each tile's allowed directions are restricted based on its parent: a tile created via a left move may only create right or forward tiles; a tile created via a right move may only create left or forward tiles; and a tile created via a forward move may create tiles in any direction, except that a subsequent left move is only permitted if the tile's most recent non-forward move was not itself a left [6]. Finally, to speed up adjacency computation during mesh generation, we construct the mesh recursively in breadth-first order, so that each tile's id corresponds to its distance from the origin.

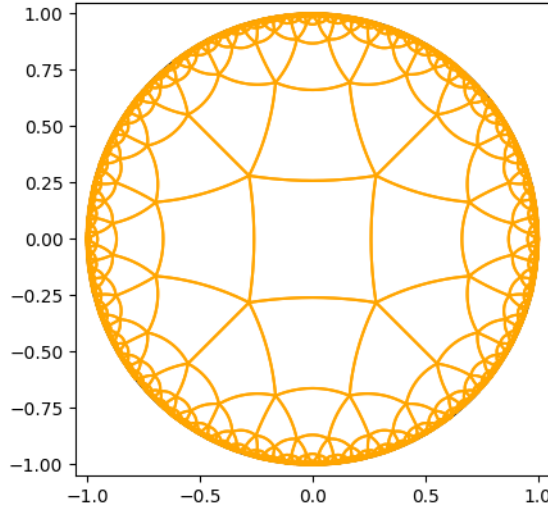


Figure 3.2: Order-5 square tiling of $\mathbb{H}^2$, viewed in the Poincaré projection.

3.2.2 Tile assignment

On a Euclidean grid, finding the tile that contains a point is trivial: divide each coordinate by the tile width and round down. In hyperbolic space this is no longer immediate. We first describe how to test whether a point lies in the *base* tile, then extend this test to an arbitrary tile, and finally show how this point-in-tile test can be used to efficiently find the tile containing a given point.

To test membership in the base tile, we map the point from the hyperboloid model, our default representation, to the gnomonic model (see Figure 3.3). The gnomonic model is the hyperbolic analogue of the gnomonic projection of the sphere: it is the central projection of the hyperboloid onto a plane, under which geodesics map to straight line segments. We construct the base tile so that, in gnomonic coordinates, it is an axis-aligned square (see Figure 3.3); membership then reduces to a pair of inequalities checking that each gnomonic coordinate of the point falls within the bounds of the square.

For a general tile, we use the fact that every tile is the image of the base tile under some isometry g of $\mathbb{H}^2$. A point p therefore lies in the tile $g \cdot T_0$ if and only if $g^{-1}p$ lies in the base tile T_0 .

Checking membership this way for every tile costs $\mathcal{O}(N_{\text{tiles}})$ per point. We reduce this to effectively constant time by exploiting the temporal locality of gradient descent: at initialization, all points lie in the base tile, and at each subsequent iteration we cache the tile assigned to each point. Provided points do not move too far between iterations, a point's new tile is either its previous tile or one of that tile's immediate neighbors, so it suffices to test only this small, fixed set rather than all N_{tiles} tiles.

3.2.3 Local tangent coordinates

We perform polynomial interpolation locally, in the tangent space at the center of each tile. The most direct way to do this on $\mathbb{H}^2$ is to map points to a tangent space using the logarithmic map, interpolate there, and map the resulting interpolation nodes back to the manifold using the exponential map. However, using a separate

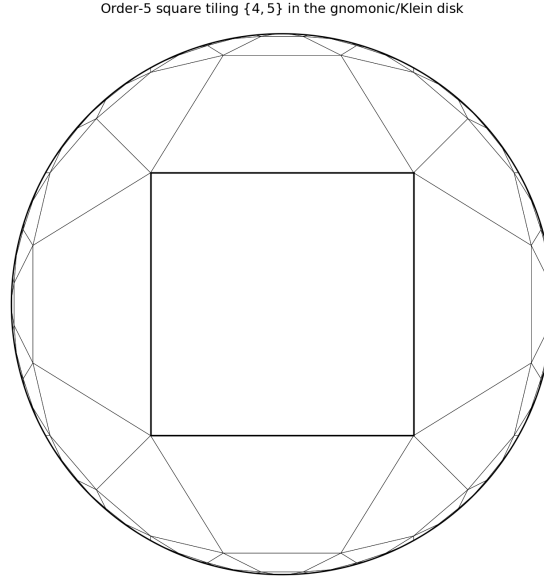


Figure 3.3: Order-5 square tiling, viewed in the gnomonic projection. The base tile is shown in the center.

tangent space for each tile leads to complicated formulas and additional implementation overhead. We therefore use a simpler tile-based construction instead: each tile is first mapped to the base tile, interpolation is then performed in the tangent space at the origin, and the resulting interpolation nodes are finally mapped back to the manifold.

Concretely, embeddings in tile t are first transformed to the base tile using the inverse of the tile’s Lorentzian transformation. The embeddings are then converted from our default hyperboloid model to the Poincaré model, since the Poincaré model has simple exponential and logarithmic maps at the origin, given by [16]:

$$\exp_0(v) = \tanh(\|v\|) \frac{v}{\|v\|}, \quad \log_0(y) = \tanh^{-1}(\|y\|) \frac{y}{\|y\|}.$$

The logarithmic map at the origin is then used to project the points onto the tangent space at the origin. In summary, the local tangent-plane coordinates $y_i^{(t)}$ for tile t are given by

$$y_i^{(t)} = \log_0 \left(\rho \left(\Lambda_t^{-1} \left(\Theta_i^{(t)} \right) \right) \right), \quad (3.1)$$

where Λ_t is the Lorentzian transformation from the base tile to tile t , ρ is the projection from the hyperboloid model to the Poincaré model, and $\Theta_i^{(t)}$ is the i -th point in tile t .

3.2.4 Interpolation grid and weights

Now that our embeddings are projected into the Euclidean tangent space, we can follow the interpolation procedure used in FIt-SNE: we construct a bounding box around the points in each tile, and within that box lay down an equispaced $p \times p$ grid of interpolation nodes. This grid serves two purposes: first, the source values are interpolated onto the grid; second, the grid points are mapped back to global coordinates so that the kernel matrix can be computed (see Figure 3.4).

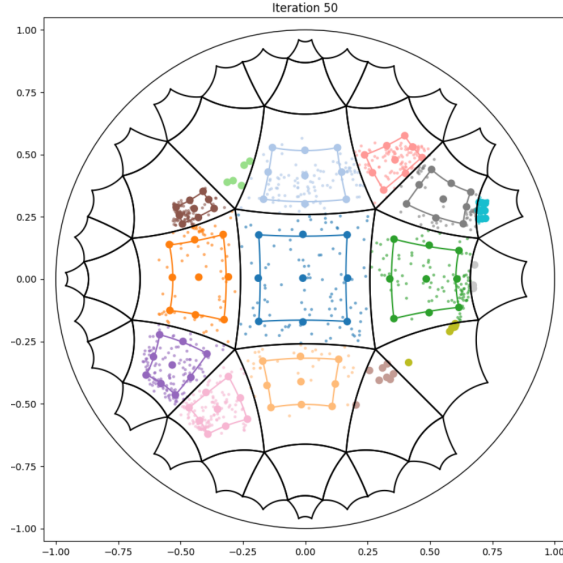


Figure 3.4: Interpolation grids on top of the datapoints, colored by tile.

We now construct this grid explicitly for a single tile t . The local coordinates of the points in tile t are

$$y_i^{(t)} = \begin{pmatrix} y_{i,1}^{(t)} \\ y_{i,2}^{(t)} \end{pmatrix} \in \mathbb{R}^2, \quad i = 1, \dots, n_t,$$

where n_t is the number of points in tile t . The bounding box of these points is given by

$$a_r^{(t)} = \min_{1 \leq i \leq n_t} y_{i,r}^{(t)}, \quad b_r^{(t)} = \max_{1 \leq i \leq n_t} y_{i,r}^{(t)}, \quad r = 1, 2.$$

The interpolation nodes in each direction are then chosen as

$$\xi_1^{(t)}, \dots, \xi_p^{(t)} \in [a_1^{(t)}, b_1^{(t)}]$$

along the x -coordinate, and

$$\eta_1^{(t)}, \dots, \eta_p^{(t)} \in [a_2^{(t)}, b_2^{(t)}]$$

along the y -coordinate.

The degree- $(p-1)$ 1D Lagrange basis in the x -coordinate is given by

$$\ell_a^{(t,1)}(x) = \prod_{\substack{r=1 \\ r \neq a}}^p \frac{x - \xi_r^{(t)}}{\xi_a^{(t)} - \xi_r^{(t)}}, \quad a = 1, \dots, p,$$

and the 1D Lagrange basis in the y -coordinate is similarly given by

$$\ell_b^{(t,2)}(y) = \prod_{\substack{r=1 \\ r \neq b}}^p \frac{y - \eta_r^{(t)}}{\eta_b^{(t)} - \eta_r^{(t)}}, \quad b = 1, \dots, p.$$

The degree- $(p-1)$ 2D Lagrange basis is the tensor product of these two 1D bases:

$$L_{ab}^{(t)}(x, y) = \ell_a^{(t,1)}(x) \ell_b^{(t,2)}(y).$$

Writing j for the flat index corresponding to the pair (a, b) , i.e. $j = a + (b - 1)p$, the interpolation matrix for the points in tile t , $B^{(t)} \in \mathbb{R}^{n_t \times p^2}$, is then given by

$$B_{ij}^{(t)} = L_{ab} \left(y_{i,1}^{(t)}, y_{i,2}^{(t)} \right)$$

The global coordinates of the interpolation nodes for tile t , written $\Gamma_j^{(t)}$, are obtained by reversing the transformations from Section 3.2.3. Writing $z_j = (\xi_a, \eta_b)$ for the local coordinates of node j (again with $j = a + (b - 1)p$), we have

$$\Gamma_j^{(t)} = \Lambda_t(\rho^{-1}(\exp_0(z_j))) .$$

3.2.5 Sparse-tile fallback

Many tiles contain fewer than $p^2 + 1$ datapoints; we refer to these as *sparse* tiles. In this case, interpolating onto a full p^2 -node grid is wasteful, since the exact points themselves can be used instead. We therefore use the raw datapoints directly as interpolation nodes, which amounts to using the identity matrix as the interpolation matrix:

$$B_{ij}^{(t)} = \delta_{ij}.$$

In the sparse-tile case, the global coordinates of the interpolation nodes are simply the original datapoints,

$$\Gamma_i^{(t)} = \Theta_i^{(t)}.$$

3.3 Efficient approximation of hyperbolic kernel sums

ϕ

The previous section showed how to construct, for each tile t , an interpolation matrix $B^{(t)}$ and a set of interpolation nodes $\Gamma^{(t)}$. We now use these to efficiently approximate the kernel sum $\phi = \mathbf{K}\mathbf{f}$ without ever forming the full $N \times N$ kernel matrix $\mathbf{K}$. Section 3.3.1 shows how interpolation gives a low-rank approximation of $\mathbf{K}$, reducing the cost of the kernel sum from $\mathcal{O}(N^2)$ to a cost governed by the much smaller number of interpolation nodes. Section 3.3.2 then corrects this approximation for nearby tiles, where interpolation is least accurate, by computing their interactions exactly.

3.3.1 Far-field approximation

Recall that the exact kernel sum is $\phi = \mathbf{K}\mathbf{f}$, where $\mathbf{K}_{ij} = K(\Theta_i, \Theta_j)$. Evaluating this directly costs $\mathcal{O}(N^2)$, since every pair of points interacts. Interpolation lets us avoid this: rather than evaluating K between all N^2 pairs of datapoints, we evaluate it only between the much smaller set of interpolation nodes, and use the interpolation matrices $\mathbf{B}^{(t)}$ to move data onto and off of this node set.

Concretely, stacking the per-tile interpolation matrices $\mathbf{B}^{(t)}$ block-diagonally gives the global interpolation matrix $\mathbf{B} \in \mathbb{R}^{N \times G}$,

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}^{(1)} & 0 & \cdots & 0 \\ 0 & \mathbf{B}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{B}^{(T)} \end{bmatrix},$$

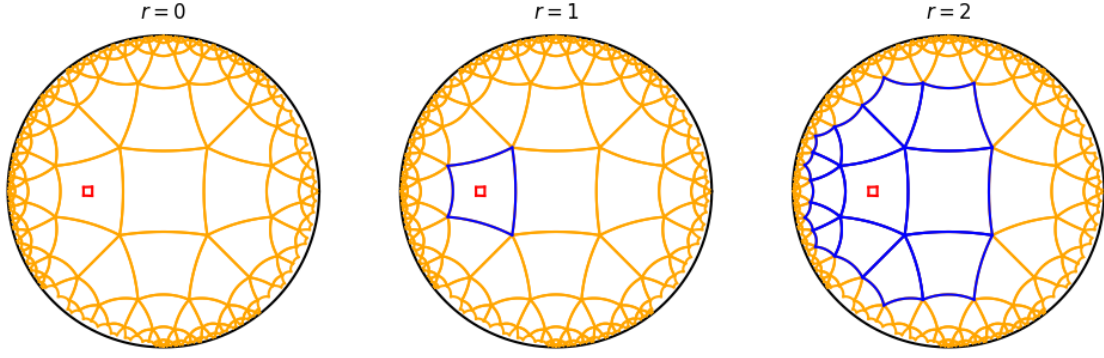


Figure 3.5: Near-field correction regions $\mathcal{N}_r(t)$ for $r = 0, 1, 2$. Tile t is marked by the red square. Blue tiles are evaluated exactly, and orange tiles are approximated by interpolation.

where $G = Tp^2$ is the total number of interpolation nodes across all T non-empty tiles. The kernel evaluated between interpolation nodes is given by the block matrix $\widetilde{\mathbf{K}} \in \mathbb{R}^{G \times G}$:

$$\widetilde{\mathbf{K}} = \begin{bmatrix} \widetilde{\mathbf{K}}^{(11)} & \widetilde{\mathbf{K}}^{(12)} & \dots & \widetilde{\mathbf{K}}^{(1T)} \\ \widetilde{\mathbf{K}}^{(21)} & \widetilde{\mathbf{K}}^{(22)} & \dots & \widetilde{\mathbf{K}}^{(2T)} \\ \vdots & \vdots & \ddots & \vdots \\ \widetilde{\mathbf{K}}^{(T1)} & \widetilde{\mathbf{K}}^{(T2)} & \dots & \widetilde{\mathbf{K}}^{(TT)} \end{bmatrix}.$$

where the block $\widetilde{\mathbf{K}}^{(ts)}$ contains the kernel evaluated between the interpolation nodes of tiles t and s ,

$$\widetilde{\mathbf{K}}_{ab}^{(ts)} = K\left(\Gamma_a^{(t)}, \Gamma_b^{(s)}\right).$$

Since, in general, $G \ll N^2$, computing $\widetilde{\mathbf{K}}$ is far cheaper than computing $\mathbf{K}$ directly. However, evaluating $\widetilde{\mathbf{K}}$ still costs $\mathcal{O}(G^2)$, rather than the $\mathcal{O}(G \log G)$ achieved by FIt-SNE. FIt-SNE reduces this cost via the FFT because two conditions hold simultaneously in the Euclidean setting (see Section 2.3.4): the kernel is translation-invariant and the interpolation nodes lie on a uniform grid, so $K_{ij} = \kappa(y_i - y_j)$ and the node-to-node kernel matrix is Toeplitz. In hyperbolic space, even though the kernel is isometry-invariant (it depends only on hyperbolic distance), the tiling nodes have no analogous linear structure: there is no indexing of the tiles under which $K(\Gamma_i, \Gamma_j)$ depends only on $i - j$. The Toeplitz structure does not arise, and the FFT cannot be applied.

Combining $\mathbf{B}$ and $\widetilde{\mathbf{K}}$, our low-rank approximation of the kernel is

$$\mathbf{K} \approx \mathbf{B} \widetilde{\mathbf{K}} \mathbf{B}^T,$$

where $\mathbf{B}^T$ first interpolates each datapoint's contribution onto the interpolation nodes, the kernel is then applied between nodes via $\widetilde{\mathbf{K}}$, and $\mathbf{B}$ interpolates the result back to the datapoints. The kernel sum ϕ can then be approximated as

$$\widetilde{\phi} = \mathbf{B} \widetilde{\mathbf{K}} \mathbf{B}^T \mathbf{f}.$$

3.3.2 Near-field correction

We cannot refine the mesh, since tile size is fixed by the topology of the tiling. Increasing the interpolation order p is also of limited use, since high-degree polynomial interpolation becomes unstable (the Runge phenomenon). With neither option

available, the only way to improve accuracy for points that are close together is to compute their interactions exactly rather than approximately. We therefore compute interactions between nearby tiles exactly, within a radius of r tiles, rather than approximating them (see Figure 3.5).

To do this, we split the kernel into a far-field part, evaluated using the low-rank approximation, and a near-field part, evaluated exactly. The far-field kernel is given by

$$\widetilde{\mathbf{K}}_{\text{far}}^{(ts)} = \begin{cases} \mathbf{0}, & \text{dist}(T_t, T_s) < r, \\ \widetilde{\mathbf{K}}^{(ts)}, & \text{otherwise,} \end{cases}$$

i.e., the approximate kernel is used for tile pairs farther than r tiles apart, and zeroed out for nearby tile pairs, since those are instead handled exactly. The corresponding near-field kernel is

$$(\mathbf{K}_{\text{near}})_{ij} = \begin{cases} K(\boldsymbol{\Theta}_i, \boldsymbol{\Theta}_j), & \text{dist}(T_{\text{tile}(i)}, T_{\text{tile}(j)}) < r, \text{ and, } i \neq j \\ 0, & \text{otherwise.} \end{cases}$$

Combining the two gives a combined kernel,

$$\mathbf{K}_{\text{comb}} \approx \mathbf{B} \widetilde{\mathbf{K}}_{\text{far}} \mathbf{B}^\top + \mathbf{K}_{\text{near}},$$

and a corrected approximation to the kernel sum,

$$\boldsymbol{\phi} \approx \mathbf{K}_{\text{comb}} \mathbf{f}.$$

In practice, we never construct the full matrix $\mathbf{K}_{\text{comb}}$ explicitly. Instead, we compute the far-field and near-field contributions separately and sum them,

$$\boldsymbol{\phi} = \boldsymbol{\phi}_{\text{far}} + \boldsymbol{\phi}_{\text{near}},$$

where

$$\boldsymbol{\phi}_{\text{far}} = \mathbf{B} \widetilde{\mathbf{K}}_{\text{far}} \mathbf{B}^\top \mathbf{f}$$

is computed using the low-rank construction from Section 3.3.1, and

$$\boldsymbol{\phi}_{\text{near}} = \mathbf{K}_{\text{near}} \mathbf{f}$$

is computed tile by tile without ever forming the full $\mathbf{K}_{\text{near}}$ matrix: for a given tile t ,

$$\boldsymbol{\phi}_{\text{near}}^{(t)} = \sum_{s=1}^T \mathbf{K}_{\text{near}}^{(ts)} \mathbf{f}^{(s)} = \sum_{s \in \mathcal{N}_r(t)} \mathbf{K}^{(ts)} \mathbf{f}^{(s)},$$

where $\mathcal{N}_r(t)$ is the set of tiles within tile-distance r of t .

This gives an efficient approximation $\boldsymbol{\phi} \approx \boldsymbol{\phi}_{\text{far}} + \boldsymbol{\phi}_{\text{near}}$ to the kernel sum, computed without ever forming the full $N \times N$ kernel matrix. In Section 3.4, we use this approximation to compute $\mathbf{Z}$ and $\mathbf{X}_i$, and assemble them into the repulsive force $\mathbf{F}_{\text{rep},i}$.

3.4 Algorithm overview and asymptotic analysis

Algorithm 1 gives an overview of the HIt-SNE computation of the kernel sums, which is used by Algorithm 2 to compute the approximate repulsive force. The asymptotic analysis in Section 3.4.2 partially addresses RQ3; the experiments in Chapter 4 provide the empirical complement.

3.4.1 Algorithm overview

Algorithm 1 Tile-interpolated computation of ϕ

Input: Datapoints Θ , interpolation order p , near-field radius r , kernel function K , source matrix F

Output: Approximate kernel sum ϕ

- 1: Assign each point in Θ to a tile
- 2: Group points by tile, writing $\Theta^{(t)}$ for the points assigned to tile t , and $F^{(t)}$ for the corresponding rows of F
- 3: **for** each nonempty tile t **do**
- 4: **if** $|\Theta^{(t)}| \leq p^2$ **then**
- 5: Use the datapoints themselves as interpolation nodes:

$$\Gamma^{(t)} \leftarrow \Theta^{(t)}, \quad B^{(t)} \leftarrow I.$$

- 6: **else**
- 7: Transform $\Theta^{(t)}$ to local tangent coordinates
- 8: Construct a $p \times p$ interpolation grid
- 9: Compute the Lagrange interpolation matrix $B^{(t)}$
- 10: Transform the interpolation nodes back to global coordinates, giving $\Gamma^{(t)}$
- 11: **end if**
- 12: **end for**
- 13: Assemble B as the block-diagonal stack of $B^{(t)}$ over all tiles t
- 14: Assemble the far-field interpolation-node kernel $\widetilde{K}_{\text{far}}$:

$$\left(\widetilde{K}_{\text{far}}^{(ts)}\right)_{ab} = \begin{cases} K\left(\Gamma_a^{(t)}, \Gamma_b^{(s)}\right), & s \notin \mathcal{N}_r(t), \\ 0, & s \in \mathcal{N}_r(t). \end{cases}$$

- 15: Compute the far-field contribution:

$$\phi_{\text{far}} \leftarrow B \widetilde{K}_{\text{far}} B^\top F.$$

- 16: Initialize $\phi_{\text{near}} \leftarrow \mathbf{0}$
- 17: **for** each nonempty tile t **do**
- 18: **for** each tile $s \in \mathcal{N}_r(t)$ **do**
- 19: Compute the exact near-field block, excluding self-interactions when $s = t$:

$$K_{ab}^{(ts)} = K\left(\Theta_a^{(t)}, \Theta_b^{(s)}\right) (1 - \delta_{ab} \delta_{ts}).$$

- 20: Add the exact contribution:

$$\phi_{\text{near}}^{(t)} \leftarrow \phi_{\text{near}}^{(t)} + K^{(ts)} F^{(s)}.$$

- 21: **end for**
 - 22: **end for**
 - 23: **return** $\phi = \phi_{\text{far}} + \phi_{\text{near}}$
-

Algorithm 2 Repulsive-force computation from kernel sums**Input:** Embedding points Θ **Output:** Repulsive force $\mathbf{F}_{\text{rep}}$

- 1: Compute $\mathbf{z} \leftarrow$ Algorithm 1 with kernel K_1 , source $\mathbf{F} = \mathbf{1}$
- 2: Set $Z \leftarrow \sum_i z_i$
- 3: Compute $\mathbf{X} \leftarrow$ Algorithm 1 with kernel K_2 , source $\mathbf{F} = \Theta$
- 4: Assemble the repulsive force:

$$\mathbf{F}_{\text{rep},i} \leftarrow \frac{1}{Z} (\Theta_i \langle \Theta_i, X_i \rangle_{n:1} + X_i).$$

5: **return** $\mathbf{F}_{\text{rep}}$ **3.4.2 Complexity**

The cost of HIT-SNE can be decomposed into the stages summarised in Table 3.1. Let N be the number of datapoints and let T be the number of nonempty tiles. For each nonempty tile t , let n_t denote the number of datapoints assigned to that tile. The sparse-tile fallback means that not every tile uses exactly p^2 interpolation nodes. Instead, the number of interpolation nodes associated with tile t is

$$m_t = \min(n_t, p^2),$$

and the total number of interpolation nodes G is given by,

$$G = \sum_{t=1}^T m_t.$$

Since $m_t = \min(n_t, p^2) \leq n_t$ and $\sum_t n_t = N$, it follows that $G \leq N$; and since $m_t \leq p^2$, we have $G \leq Tp^2$. In practice, we restrict our embeddings to a fixed tiling with T_{tiling} tiles, so $G \leq T_{\text{tiling}} \cdot p^2$.

Per-stage runtime

Stage	Runtime
Tile assignment	$\mathcal{O}(N)$
Coordinate transformation	$\mathcal{O}(N)$
Spread: datapoints $\rightarrow$ nodes	$\mathcal{O}(Np^2)$
Far-field kernel evaluation	$\mathcal{O}(G^2)$
Far-field kernel application	$\mathcal{O}(G^2)$
Gather: nodes $\rightarrow$ datapoints	$\mathcal{O}(Np^2)$
Near-field correction	$\mathcal{O}\left(\sum_t \sum_{s \in \mathcal{N}_r(t)} n_t n_s\right)$
Repulsive force computation	$\mathcal{O}(N)$

Table 3.1: Per-stage runtime cost of HIT-SNE.

Tile assignment. Naively, assigning a datapoint to a tile could require testing against every tile, giving a cost proportional to the total number of tiles. However, as discussed in Section 3.2.2, the implementation reuses the tile assignment from the previous iteration. Since datapoints move only a small amount between iterations, it is usually sufficient to check the previous tile and the 12 neighboring tiles, resulting in a constant amount of time per point lookup. Since there are N points the tile assignment stage runtime is $\mathcal{O}(N)$.

Coordinate transformation, spread, and gather. Each datapoint is transformed into the coordinate system of its tile and projected to a local tangent space. This requires a constant amount of work per datapoint, so the coordinate transformation step has runtime $\mathcal{O}(N)$. Datapoints in non-sparse tiles are then interpolated onto the interpolation nodes via a matrix-vector multiply with the block-diagonal matrix $\mathbf{B}^T$, which has at most Np^2 non-zero entries, giving a spread cost of $\mathcal{O}(Np^2)$. The symmetric gather step (interpolating node values back to datapoints via $\mathbf{B}$) has the same cost.

Far-field kernel evaluation and application. The far-field kernel matrix is evaluated between the G interpolation nodes, giving a dense $G \times G$ matrix at cost $\mathcal{O}(G^2)$. Applying this matrix to the interpolated source values costs a further $\mathcal{O}(G^2)$.

Near-field correction. The near-field correction computes exact interactions between datapoints in nearby tiles, at cost

$$\mathcal{O}\left(\sum_{t=1}^T \sum_{s \in \mathcal{N}_r(t)} n_t n_s\right).$$

The near-field radius r controls the size of this term: at $r = 0$ the sum is empty and this stage costs nothing; increasing r adds more tile pairs to each inner sum, raising the near-field cost while improving accuracy.

Repulsive force computation. Once ϕ has been assembled from the far-field and near-field contributions, the repulsive force on each datapoint is computed from ϕ and the normalisation constant $Z = \sum_i \phi_i$. Both require a single pass over the N datapoints, giving a cost of $\mathcal{O}(N)$.

Total runtime

Combining the stages, the total runtime is

$$\mathcal{O}\left(Np^2 + G^2 + \sum_{t=1}^T \sum_{s \in \mathcal{N}_r(t)} n_t n_s\right).$$

In the $r = 0$ case the near-field term vanishes; substituting the bound $G \leq T_{\text{tiling}} \cdot p^2$ for a fixed tiling gives $\mathcal{O}(Np^2 + (T_{\text{tiling}} \cdot p^2)^2)$. Disregarding p , as $p = 3$ is typically used, the second term is constant for a fixed tiling, so the overall cost is $\mathcal{O}(N)$.

In the $r = 1$ and $r = 2$ case, the analysis depends on how points are distributed across tiles. Even under a uniform distribution with $n_t \approx N/T$, the near-field sum evaluates to $\mathcal{O}(T \cdot |\mathcal{N}_r| \cdot (N/T)^2) = \mathcal{O}(N^2/T)$, which remains superlinear in N unless T grows with N . In the worst case, during the early exaggeration phase, points cluster in the base tile and the algorithm devolves to the exact method's runtime of $\mathcal{O}(N^2)$. In Chapter 6, we will discuss possible solutions to this issue.

Memory

In practice, neither the far-field nor the near-field matrices need to be stored, since only the matrix-vector product is required. The interpolation matrix $\mathbf{B}$ is stored, as it is used twice per iteration (once for the spread and once for the gather), but it is block-diagonal with at most Np^2 nonzero entries. Furthermore, since $\mathbf{B}$ could be easily recomputed instead of being stored we could also calculate the spread and gather step in $\mathcal{O}(N)$. The theoretical memory requirement is therefore $\mathcal{O}(N)$.

Chapter 4

Experiments

This chapter describes the experiments conducted to evaluate HIt-SNE against the exact method, addressing the empirical component of RQ3. Section 4.1 explains the datasets with which the experiment will be conducted. Section 4.2 describes the experimental setup, including the dataset, hardware and software used, and the hyperparameters shared across all methods. Section 4.3 describes the first experiment, which examines how runtime and accuracy change at different iterations of the embedding trajectory. Section 4.4 describes the second experiment, which examines how runtime and accuracy scale with the number of embedded points. Section 4.5 describes the third experiment, in which the final embeddings are computed entirely in that method as would be done in the real-world setting, and the visual quality of the output will be compared.

4.1 Datasets

In our experiments we used two different datasets: MNIST and Planaria.

The MNIST dataset consists of 70,000 grayscale images of handwritten digits, each of size 28×28 pixels, giving a native dimensionality of 784 [26]. Each image is labeled with the digit it depicts, from 0 to 9, giving 10 classes of roughly 7,000 images each (see Figure 2.2). MNIST is a standard benchmark for dimensionality reduction and visualization algorithms: its known class structure makes it easy to assess whether an embedding has preserved meaningful groupings, since a good embedding should produce visually distinct clusters corresponding to each digit class. It is worth noting, however, that MNIST does not have hierarchical structure in the sense that motivates hyperbolic t-SNE since the ten digit classes have no natural tree relation between them. It is used here as a benchmark because it is well-understood and widely used in the t-SNE literature, not because it is a natural fit for hyperbolic embedding.

The Planaria dataset consists of about 2000 high-dimensional vectors, each recording the gene expression counts of a single cell of a specimen of *Schmidtea mediterranea*, a planarian, a type of flatworm studied for its immortality [39]. The hierarchical structure of the Planaria dataset can be seen in Figure 4.1 as a cell lineage: cells begin as neoblasts, a type of stem cell, and get differentiated further and further until they become the familiar skin and muscle cells [39]. We included the Planaria dataset in our experiments due to its hierarchical structure, as it is typical of the data one would want to use hyperbolic t-SNE for.

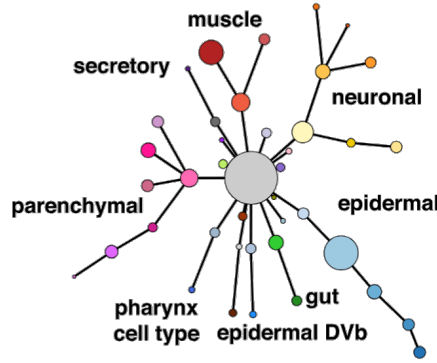


Figure 4.1: Illustration from [39]. Lineage tree of Planaria cells, showcasing hierarchical structure of cell types.

4.2 Setup

Experiments were conducted on the MNIST dataset [26] and the Planaria dataset [39]. They were run on a single machine with an Intel Core Ultra 7 155H CPU, 16 physical cores, 22 logical CPUs, and 30 GiB of RAM. The code for tiling generation and visualization was written in a separate C++ program. All other programs were implemented in Python using NumPy v1.26.

To produce a t-SNE or hyperbolic t-SNE embedding, one starts with a PCA embedding of sampled data from the dataset and then performs 2000 iterations of gradient descent. Let us refer to the embedding at iteration k as $\Theta^{(k)}$. We define an *embedding trajectory* as the set of embeddings at all iterations:

$$\{\Theta^{(k)} \mid k \in [0, 2000)\}.$$

To generate the ground-truth embedding trajectories, the exact hyperbolic t-SNE method was run with the following hyperparameters: perplexity = 30, number of iterations = 2000, and early exaggeration factor = 12 applied for the first 250 iterations. For experiments 1 and 2, a learning rate of 200 was used, for experiment 3 a learning rate of 10 was used for MNIST, and 2 for Planaria. Each embedding trajectory was generated by sampling datapoints without replacement from the MNIST dataset using a specified random seed.

HIt-SNE additionally requires a one-time setup cost to construct its tiling and corresponding 1-nearest neighbor graph. The tiling was generated using the C++ visualization tool so that it encompassed all points appearing throughout the embedding trajectory, resulting in a depth-7 tiling containing 2941 tiles. The neighbor graph was subsequently generated in Python from the tiling.

In our experiments, we chose to measure the error and runtime associated with computing the normalizing constant Z rather than the full gradient. The normalizing constant is simpler to analyse, being a single scalar quantity, while still requiring the same kernel evaluation as the full gradient computation.

We compare four methods throughout: the exact method and HIt-SNE with near-field radii $r = 0, 1, 2$. At $r = 0$, no near-field correction is applied; at $r = 1$, interactions between points within the same tile are computed exactly; and at $r = 2$, interactions between points in the same tile and adjacent tiles are computed exactly.

We fix $p = 3$ as is done in the original FIt-SNE paper for all trials [27]. For each method, we measure the wall-clock time required to compute Z and the relative error of the approximate value of Z compared to the exact value.

4.3 Experiment 1: Per-iteration runtime and error

The goal of this experiment is to verify two expected properties of HIt-SNE during a single embedding trajectory. First, at every iteration, the more approximate methods (lower r) should achieve lower runtimes than the more exact methods (higher r and the exact method). Second, this speed advantage should come at the cost of reduced accuracy, while the resulting error remains within a reasonable range throughout the embedding process.

To test this, we generated a single embedding trajectory using the exact method, consisting of 50k MNIST points. We did three separate runs. For each run we randomly selected a set of 5k points from the 50k embedding. At each iteration, we computed both the exact value of Z and the HIt-SNE approximation of Z for $r = 0, 1, 2$, recording the runtime and relative error of each method.

We expect that for the $r = 0$ and the exact method, the runtime curve should be constant, as both methods should be agnostic to the embedding point distribution (which clustered densely in the base tile until around iteration 300). On the other hand we expect that the runtime of the $r = 1$ method should be the same as the exact method until around iteration 300 when points begin to leave the base tile, and then it should decrease as the embeddings are more evenly spaced throughout the disk. Likewise, for $r = 2$ the method should be the same as the exact method until the points leave the inner 2 rings of points, and then should decrease in further iterations. As for the error, we expect that the $r = 0$ method should decrease in error, since at later iterations far-field interactions dominate.

4.4 Experiment 2: Runtime and error scaling with problem size

The goal of this experiment is to examine how runtime and error scale with the number of embedded points N . Since the exact method computes pairwise interactions between all points, its runtime is expected to scale quadratically, $\mathcal{O}(N^2)$. HIt-SNE, by contrast, is expected to scale linearly, $\mathcal{O}(N)$, for $r = 0$. HIt-SNE for $r = 1$ and $r = 2$ should also scale linearly when datapoints are somewhat evenly distributed among the tiles.

To test this, we generated a single embedding trajectory consisting of 50k MNIST points (using the exact method) and selected iteration 1990, where the points appeared to be well distributed throughout the embedding space. The problem sizes N are logarithmically spaced between 10^3 and 5×10^4 ,

$$N \in \{1000, \dots, 50000\} \quad (\text{logarithmically spaced}). \quad (4.1)$$

For each N , we generated 10 subsets of that size without replacement from the full 50k-point set. For each sampled subset, we evaluated all four methods (the exact method and HIt-SNE with $r = 0, 1, 2$), recording the runtime and relative error. The mean and standard deviation were then computed across all subsets of the same size.

We expect that the exact method will scale quadratically since all $\mathcal{O}(N^2)$ interactions are computed. We expect that the $r = 0, 1, 2$ methods will scale approximately linearly, with $r = 0$ being the fastest, followed by $r = 1$ and $r = 2$. As for error, we expect $r = 0$ to have the highest error as no near-field correction is added, followed by $r = 1$ and $r = 2$.

4.5 Experiment 3: Visual Fidelity of the Final Embedding

In this experiment we aim to determine how the HIT-SNE accelerated final embeddings visually compare to the exact method. Here we sampled a subset of 1000 samples from both the MNIST dataset and the Planaria dataset and generated embeddings using all four methods using 2000 iterations of gradient descent. At the end we visually compare the results for relative cluster position and fidelity to the exact method.

We expect that $r = 1$ and $r = 2$ should have the same relative cluster structure as the exact method, as clusters are formed in the early exaggeration phase which near-field correction should accurately capture. For $r = 0$ it is less clear whether the relative clustering structure should hold as there is no near-field correction so the early exaggeration phase may produce different clusters and in different positions.

Chapter 5

Results

This chapter presents the results of the three experiments described in Chapter 4. Section 5.1 presents the results of Experiment 1, examining runtime and accuracy over the course of a single embedding trajectory. Section 5.2 presents the results of Experiment 2, examining how runtime and accuracy scale with the number of embedded points. Section 5.3 presents the results of Experiment 3, showing the difference in visual quality between the methods.

5.1 Experiment 1: Per-iteration runtime and error

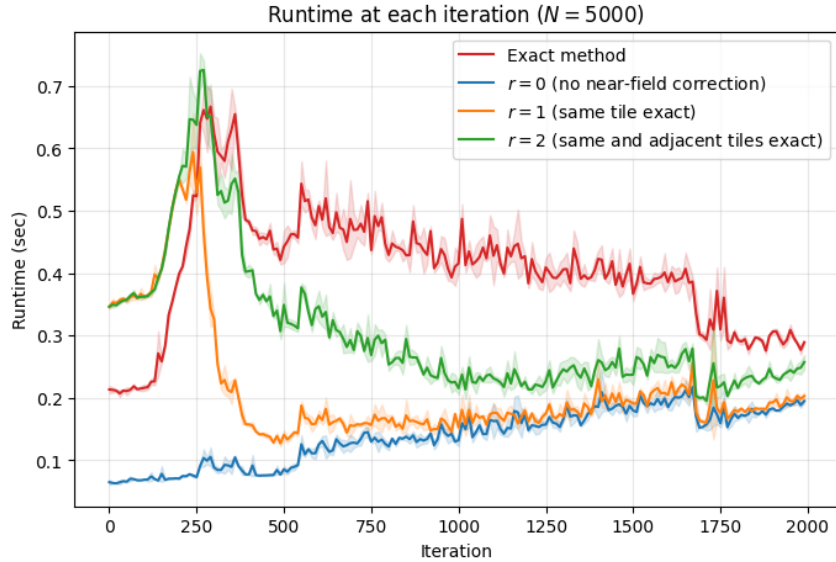


Figure 5.1: Mean runtime per iteration during a single embedding trajectory, for the exact method and HIt-SNE with near-field radius $r = 0, 1, 2$. Results are averaged across three embedding trajectories; shaded regions show one standard deviation.

Figure 5.1 shows the mean runtime per iteration for each method. The runtime for all methods except for $r = 0$ increases sharply near the beginning (around iteration 300) and then decreases afterwards. The relative runtime ordering between the methods is as expected with $r = 2$ the slowest, then $r = 1$, and finally $r = 0$. All methods, except for $r = 0$, show an initial rise in runtime over the first few hundred iterations.

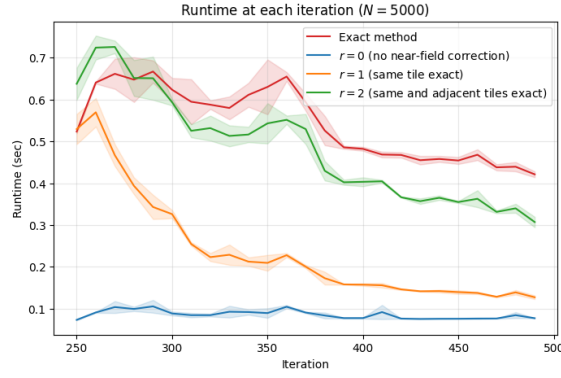


Figure 5.2: A zoomed-in version of Figure 5.1, with only iterations 250 to 500 shown, to highlight the critical period after early exaggeration.

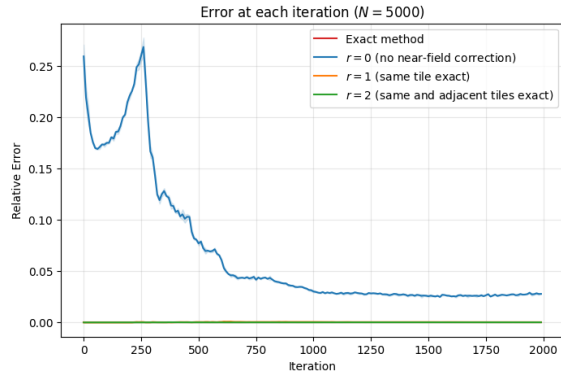


Figure 5.3: Mean relative error per iteration between the HIt-SNE relative error in computing Z and the exact Z , for $r = 0, 1, 2$, averaged across three embedding trajectories. Shaded regions show one standard deviation.

Figure 5.3 shows the mean relative error per iteration for the three HIt-SNE variants. For $r = 0$ the error is high during early exaggeration, and then begins to monotonically decrease. For $r = 1$ and $r = 2$, relative error is negligible throughout.

5.2 Experiment 2: Runtime and error scaling with problem size

Figure 5.4 shows mean runtime as a function of problem size N . The exact method’s runtime increases quadratically as N grows from 1k to 50k. In contrast, all three HIt-SNE variants show a much flatter runtime curve across the same range of N , remaining close together and growing only modestly as N increases, with $r = 2$ slightly higher than $r = 1$, which is in turn slightly higher than $r = 0$.

Figure 5.5 shows mean relative error as a function of problem size N , at the same fixed iteration. Across all problem sizes, the ordering between variants is consistent, with $r = 0$ showing the highest error, followed by $r = 1$, then $r = 2$ with the lowest. For the $r = 0$ method the error remains relatively constant. For $r = 1$ the error seems to gradually increase and then asymptote. For $r = 2$ the error tracks $r = 1$ then around a problem size of about 7k it begins to monotonically decrease.

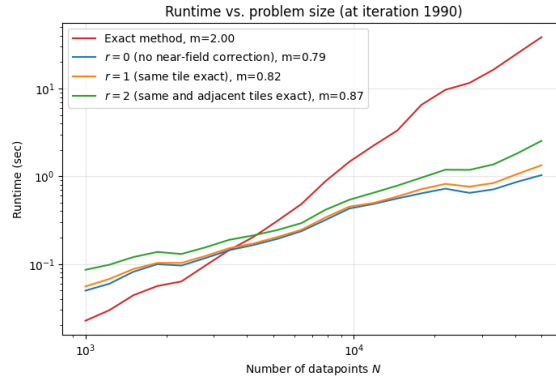


Figure 5.4: Mean runtime as a function of problem size N , evaluated at iteration 1990 of a 50k-point embedding trajectory, for the exact method and HIt-SNE with $r = 0, 1, 2$. Results are averaged across repeated runs per problem size. Standard deviations are not shown as they are too small to be visible. The linear regression slope m is given in the legend.

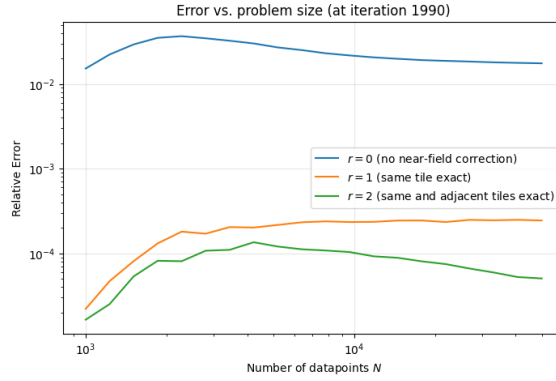


Figure 5.5: Mean relative error as a function of problem size N , evaluated at iteration 1990, for HIt-SNE with $r = 0, 1, 2$. Standard deviation is not shown as it is too small to be visible.

5.3 Experiment 3: Visual Fidelity of the Final Embedding

Figure 5.6 shows the final embeddings for the exact, and approximate methods for a subset of the MNIST dataset. HIt-SNE with $r = 2$ produces visually identical results to the exact method. HIt-SNE with $r = 1$ produces almost identical results to the exact method, however some of the individual points are slightly moved (see the green cluster in the northeast), but the relative clustering remains the same. $r = 0$ produces an embedding where points are concentrated near the boundary of the disk, but the relative cluster positions are somewhat preserved (see Figure 5.7).

Figure 5.8 shows the final embeddings for a subset of the Planaria dataset. These embeddings share with the MNIST embeddings that $r = 1$ and $r = 2$ resemble the exact method, but $r = 0$ is markedly different. The relative clustering in the Planaria dataset is harder to ascertain, but there tends to be a consistency in the neuronal class being central, and the neoblast class being located on the outskirts.

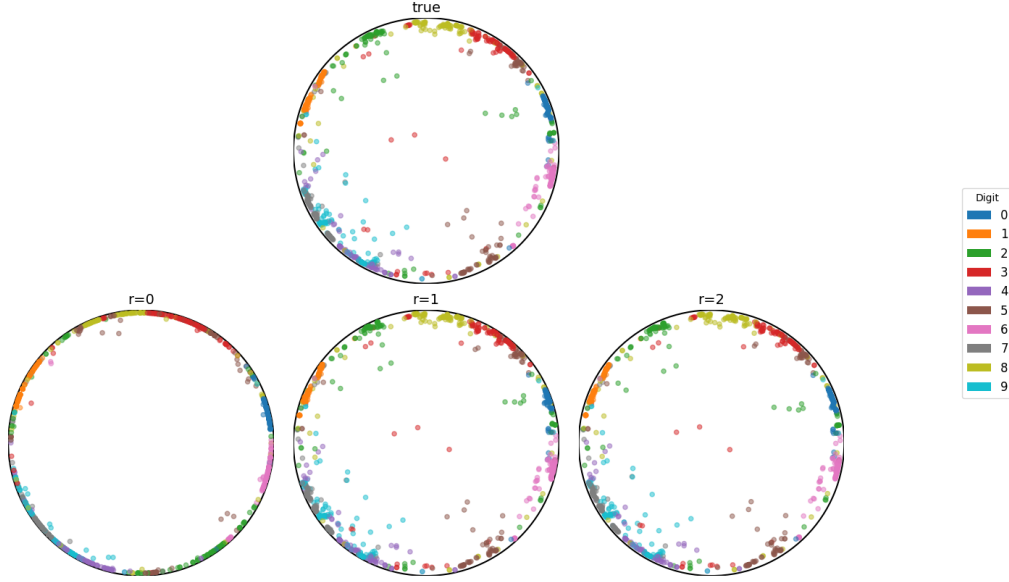


Figure 5.6: Final embeddings generated on a 1k sample of MNIST using the exact (true) method as compared to approximate methods ($r = 0, 1, 2$). The colors correspond to the MNIST class labels (i.e the digits 0-9).



Figure 5.7: A plot of the angle of each embedded point on the Poincaré disk, for each method, on a 1k subset of MNIST. This plot is an unrolled version of the embedding, and helps show the positions of the clusters relative to each other. The colors correspond to the MNIST class labels (i.e the digits 0-9).

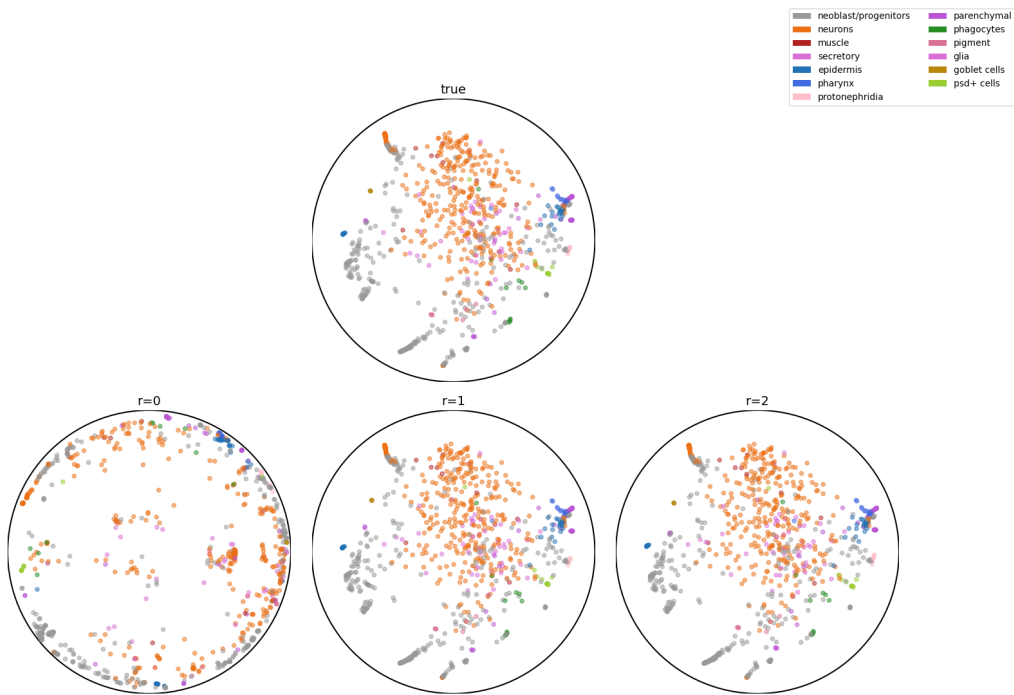


Figure 5.8: Final embeddings generated on a 1k sample of the Planaria dataset using the exact (true) method as compared to approximate methods ($r = 0, 1, 2$). The colors correspond to classes of cell types.

Chapter 6

Discussion

In this chapter we discuss the implications of our results and the limitations of the HIt-SNE algorithm and our experiments. The sections below constitute our answer to RQ3: *How does HIt-SNE perform relative to the exact method in terms of runtime, approximation accuracy, and visual fidelity of the resulting embeddings?*

6.1 Runtime Behavior

We found empirically (see Figure 5.4) that all variants of HIt-SNE outperform the exact method for sufficiently large problem sizes, with the exact method scaling approximately quadratically and HIt-SNE variants scaling approximately linearly, consistent with the asymptotic analysis in Section 3.4.2. Two observations from the experiments complicate this picture.

The first complication arises in Experiment 1. The runtime of the exact method varies across iterations similarly to the $r = 1$ and $r = 2$ variants (see Figure 5.1), which is unexpected: each exact iteration requires the same N^2 interactions regardless of where the points are. Multiple runs reproduced this behavior. Since the deviation is less than an order of magnitude and the main qualitative result, that all HIt-SNE variants are faster than the exact method after the early exaggeration phase, is unaffected, we do not investigate this further.

The second complication, also in Experiment 1, concerns the runtime of the $r = 1$ and $r = 2$ variants during early exaggeration. During the first ~ 300 iterations, all points are concentrated in the base tile, so the near-field correction should dominate and runtime should resemble the exact method’s and be constant. Instead, the measured runtime starts low and rises. One possible explanation is that at $N = 5000$ the problem fits in CPU cache and memory-access patterns change as points spread out. This remains an open question. Note that this behavior is separate from the deeper performance problem during early exaggeration — the $\mathcal{O}(N^2)$ cost when all points are concentrated in a single tile — which is discussed in Section 6.3.

6.2 Approximation Quality

We found empirically that the $r = 1$ and $r = 2$ modes produce embeddings that are visually true to the exact method, both on the MNIST digit dataset and on the Planaria dataset (see Section 5.3). The $r = 1$ method shows very slight deviations

in the position of individual points, but the overall cluster structure is preserved.

The $r = 0$ method, by contrast, accumulates non-negligible error at every iteration. Its error profile is consistent with theoretical expectations: error is highest during early exaggeration, when all points are concentrated in the base tile and near-field interactions dominate, and decreases as points spread across tiles. One empirical drawback of $r = 0$ is reduced stability during early exaggeration, requiring a smaller learning rate to avoid large gradients. A second drawback is that $r = 0$ embeddings tend to crowd toward the boundary of the Poincaré disk, producing less well-separated clusters.

6.3 The Early Exaggeration Problem and the Hybrid Method

The most significant practical limitation of Hit-SNE is that the accurate modes ($r = 1$ and $r = 2$) effectively devolve to the exact method during the early exaggeration phase. As noted in Section 3.4.2, when all points concentrate in the base tile the near-field correction must cover every pair of points, giving an $\mathcal{O}(N^2)$ cost per iteration. This prevents running $r = 1$ or $r = 2$ on large datasets where the exact method is already too slow.

A practical workaround is a hybrid method: use $r = 0$ during early exaggeration (or until points are sufficiently spread across tiles), then switch to $r = 1$ for the remainder of the optimization. A comparison of the hybrid method against $r = 0$ and the exact method is shown in Figure 6.1. The hybrid embedding preserves the cluster structure of the $r = 0$ method while avoiding the boundary crowding that $r = 0$ exhibits, producing embeddings closer in character to those of the exact method.

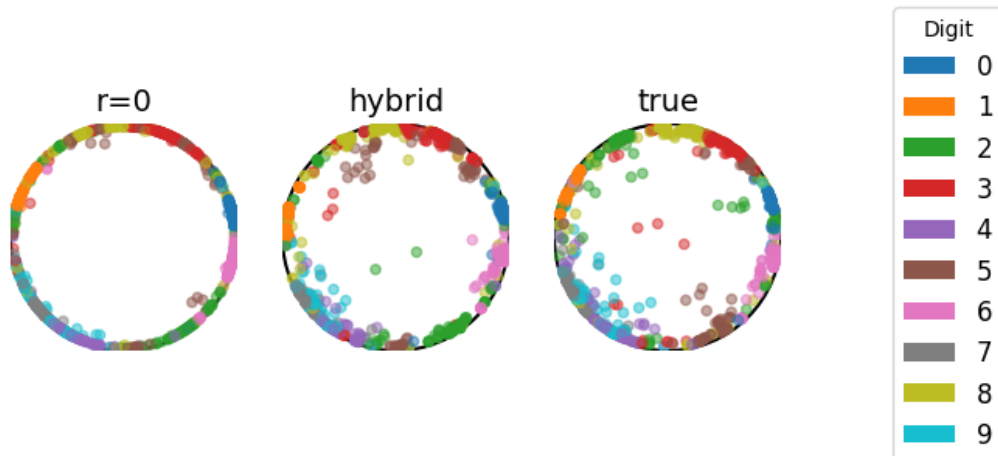


Figure 6.1: Final embeddings generated on a 1k sample of MNIST using the exact (true) method, the $r = 0$ method, and the hybrid method. Colors correspond to MNIST class labels (digits 0–9).

6.4 Limitations

Several aspects of the algorithm and the experiments remain untested or constrained.

We did not vary the interpolation order p . The FIt-SNE paper [27] recommends $p = 3$ and cautions against higher values due to the Runge phenomenon. Preliminary experiments using higher p and Chebyshev interpolation nodes — which mitigate the Runge phenomenon — did not produce a meaningful reduction in error, so we fixed $p = 3$ throughout.

We did not investigate the choice of tiling depth. For our experiments we selected a tiling size by hand and constrained all points to lie within a fixed radius on the Poincaré disk. Choosing too small a radius introduces distortion as points cluster near the edge of the disk instead of spreading out. A practical implementation would benefit from an adaptive tiling scheme, such as the one implemented in *HyperRogue* [24], that expands the tiling dynamically as points spread out.

The scaling experiment (Experiment 2) was limited to a single late iteration when points are most uniformly distributed, which presents the best-case scenario for the near-field term and does not reflect the harder early-exaggeration regime. Scaling plots at various iterations could be added to assess the impact of the near-field correction on runtime.

Chapter 7

Conclusion

This chapter concludes the thesis. Section 7.1 returns to the research question and its sub-questions, summarizing what this thesis has shown and where its results remain incomplete. Section 7.2 then looks beyond this thesis, discussing directions for future research that follow from these results and their limitations.

7.1 Answering the Research Question

This thesis set out to answer whether the interpolation-based acceleration strategy of FIt-SNE could be extended to hyperbolic t-SNE. We answer this question affirmatively. HIt-SNE, the method proposed in this thesis, shows that the same core idea behind FIt-SNE’s success carries over to hyperbolic space. In Euclidean space, FIt-SNE works by replacing exact pairwise computation with local polynomial interpolation. This idea still works once the gradient, tiling, and coordinate system are suitably adapted for hyperbolic geometry. The result is a method that is substantially faster than the exact approach, while remaining accurate enough for practical use.

To answer our first sub-question: the hyperbolic t-SNE gradient can indeed be reformulated and decomposed into components suitable for interpolation-based approximation. First, we rewrote the gradient in the hyperboloid model, rather than the Poincaré model used in existing implementations. We then verified that this hyperboloid gradient agrees with the original Poincaré-model gradient. Second, we rewrote this hyperboloid gradient as a small set of kernel sums, mirroring the decomposition used by FIt-SNE in Euclidean space. This non-trivial decomposition is what makes the gradient amenable to interpolation-based approximation in the first place.

To answer our second sub-question: a regular hyperbolic tiling, combined with a tile-local tangent-space coordinate scheme, makes accurate polynomial interpolation possible on hyperbolic space. This is true even though the tiling itself cannot be refined the way a Euclidean grid can. We built this tiling using the construction of [6]. To assign points to tiles efficiently, we used the gnomonic model, in which tile membership reduces to a simple bounding-box test, and combined this with caching to exploit the fact that points move only a small distance between iterations. Within each tile, we performed interpolation locally in tangent-space coordinates, and for tiles with too few points to fill a full interpolation grid, we fell back to using their datapoints directly as interpolation nodes. To recover the accuracy lost from being

unable to refine the tiling, we split the kernel sum into a near-field part, computed exactly, and a far-field part, computed via interpolation. We efficiently computed the near-field part tile by tile, without ever constructing the full near-field kernel matrix, keeping this correction efficient even as the near-field radius grows.

To answer our third sub-question: the runtime of HIt-SNE scales favorably relative to the exact method, though our results here leave a few open questions. Its accuracy also remains within a practical range, with some complications as well. Our measured runtimes were broadly consistent with the expected asymptotic behavior, and all HIt-SNE variants were substantially faster than the exact method throughout. For both $r = 1$ and $r = 2$, error remained low enough for practical use at every iteration tested, and error also stayed stable as the number of embedded points increased, with no sign of degrading at larger problem sizes. The hybrid method proposed in Section 6.3 also addresses concerns with the tradeoff between runtime and accuracy.

Overall, in terms of operating advice, it seems that the $r = 1$ option is superior to the exact method and can serve as a drop-in replacement. The hybrid method, however, handles large datasets well but produces embeddings that differ slightly from the exact approach.

7.2 Future Research Directions

Here we specify a few possible directions for future research.

One useful direction would be to compare HIt-SNE to the only other acceleration scheme for hyperbolic t-SNE: the Barnes-Hut-based acceleration of Skrodzki et al. [44]. This would provide practical guidelines on when to use HIt-SNE over the Barnes-Hut method. It would also be useful to evaluate the hybrid method and the $r = 0$ method using the embedding distortion metrics presented in that work, namely precision, recall, and nearest neighbor error, to determine whether, despite having a different relative cluster structure from the exact method, they remain equally informative about the underlying data.

Another area for further research would be to extend the recently developed Fast Multipole Method of [14] into an acceleration scheme for hyperbolic space, as the FMM offers better error bounds than either HIt-SNE or Barnes-Hut.

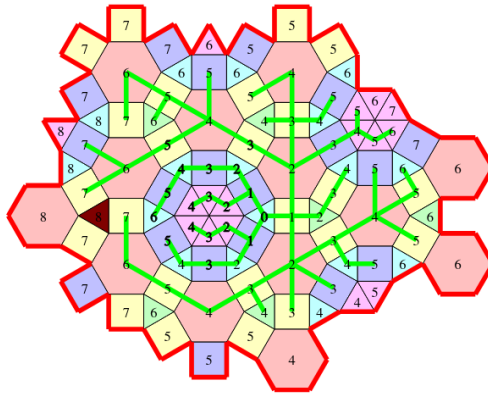


Figure 7.1: Illustration of a non-regular tiling from [10].

Additionally, since tile size in hyperbolic space is determined by the geometry of

the tiling, specifically the number of tiles meeting at each vertex and their angles, tile refinement could potentially be achieved by switching to a different regular tiling or a non-regular tiling (see Figure 7.1). This might reduce the need for the hybrid method and allow higher accuracy at lower cost during the early exaggeration period.

Further experiments on additional datasets, including highly hierarchical ones evaluated with relevant quality metrics, could provide a useful indication of the method's robustness. Similarly, running Experiment 2 at different iterations of the process and Experiment 1 with larger N values could help clarify the early exaggeration problem and the anomalous behavior of the exact method.

Finally, gradient descent with momentum, as used in the Barnes-Hut method of Skrodzki et al., could be adopted to speed up the optimization. Going a step further, stochastic gradient descent could also be explored.

Bibliography

- [1] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ: Princeton University Press, 2008. ISBN: 978-0-691-13298-3.
- [2] T. S Andrews, V. Y Kiselev, and M. Hemberg. “Tutorial: Guidelines for the computational analysis of single-cell rna sequencing data”. In: *Nature Protoc.* (2021).
- [3] Josh Barnes and Piet Hut. “A hierarchical $O(N \log N)$ force-calculation algorithm”. en. In: *Nature* 324.6096 (Dec. 1986), pp. 446–449. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/324446a0. URL: <https://www.nature.com/articles/324446a0> (visited on 07/19/2026).
- [4] Mikhail Belkin and Partha Niyogi. “Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering”. en. In: *Advances in Neural Information Processing Systems 14*. Ed. by Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani. The MIT Press, Nov. 2002, pp. 585–592. ISBN: 978-0-262-27173-8. DOI: 10.7551/mitpress/1120.003.0080. URL: <https://direct.mit.edu/books/book/2485/chapter/66411/Laplacian-Eigenmaps-and-Spectral-Techniques-for> (visited on 06/24/2026).
- [5] Jan Niklas Böhm, Philipp Berens, and Dmitry Kobak. “Attraction-Repulsion Spectrum in Neighbor Embeddings”. en. In: *Journal of Machine Learning Research* 23 (Mar. 2022).
- [6] R.P.G van Buren. “Navigational Ability in Hyperbolic Space: A Study in VR”. en. MA thesis. Delft University of Technology, 2025.
- [7] Gaëlle Candel and David Naccache. *Index t -SNE: Tracking Dynamics of High-Dimensional Datasets with Coherent Embeddings*. en. arXiv:2109.10538 [cs.LG]. Sept. 2021. DOI: 10.48550/arXiv.2109.10538. URL: <http://arxiv.org/abs/2109.10538> (visited on 06/24/2026).
- [8] James W. Cannon et al. “Hyperbolic Geometry”. In: *Flavors of Geometry*. Ed. by Silvio Levy. Vol. 31. MSRI Publications. Cambridge University Press, 1997, pp. 59–115.
- [9] Xiaofeng Cao. “Poincaré Fréchet mean”. en. In: *Pattern Recognition* 137 (May 2023), p. 109302. ISSN: 00313203. DOI: 10.1016/j.patcog.2023.109302. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0031320323000031> (visited on 07/23/2026).

- [10] Dorota Celińska-Kopczyńska and Eryk Kopczyński. *Generating Tree Structures for Hyperbolic Tessellations*. en. arXiv:2111.12040 [cs]. Mar. 2022. DOI: 10.48550/arXiv.2111.12040. URL: <http://arxiv.org/abs/2111.12040> (visited on 05/06/2026).
- [11] Arman Cohan et al. “SPECTER: Document-level Representation Learning using Citation-informed Transformers”. en. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020, pp. 2270–2282. DOI: 10.18653/v1/2020.acl-main.207. URL: <https://www.aclweb.org/anthology/2020.acl-main.207> (visited on 02/27/2026).
- [12] Xiran Fan, Chun-Hao Yang, and Baba C. Vemuri. *Nested Hyperbolic Spaces for Dimensionality Reduction and Hyperbolic NN Design*. en. arXiv:2112.03402 [cs.LG]. Dec. 2021. DOI: 10.48550/arXiv.2112.03402. URL: <http://arxiv.org/abs/2112.03402> (visited on 07/22/2026).
- [13] Christiane Fellbaum. *WordNet: An electronic lexical database*. MIT press, 1998.
- [14] Lucas Frans Willem Furer. “FMM-t-SNE: Accelerating t-SNE with the fast multipole method”. en. MA thesis. Delft University of Technology, Mar. 2026.
- [15] Jean Gallier and Jocelyn Quaintance. *Differential Geometry and Lie Groups: A Computational Perspective*. en. Vol. 12. Geometry and Computing. Cham: Springer International Publishing, 2020. ISBN: 978-3-030-46039-6 978-3-030-46040-2. DOI: 10.1007/978-3-030-46040-2. URL: <http://link.springer.com/10.1007/978-3-030-46040-2> (visited on 07/22/2026).
- [16] Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. “Hyperbolic Neural Networks”. en. In: *Advances in Neural Information Processing Systems* 31 (2018).
- [17] *GeoGebra Constructions in the Poincaré Disc*. URL: <https://www.malinc.se/noneuclidean/en/poincaredisc.php>.
- [18] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. eng. 4. ed. Johns Hopkins studies in the mathematical sciences. Baltimore, MD: Johns Hopkins Univ. Press, 2013. ISBN: 978-1-4214-0794-4.
- [19] Caglar Gulcehre et al. “Hyperbolic Attention Networks”. en. In: *7th International Conference on Learning Representations*. New Orleans, LA, USA.
- [20] Yunhui Guo, Haoran Guo, and Stella X. Yu. “CO-SNE: Dimensionality Reduction and Visualization for Hyperbolic Data”. en. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA: IEEE, June 2022, pp. 11–20. ISBN: 978-1-6654-6946-3. DOI: 10.1109/CVPR52688.2022.00011. URL: <https://ieeexplore.ieee.org/document/9879850/> (visited on 11/11/2025).
- [21] Daniel Jiwoong Im, Nakul Verma, and Kristin Branson. *Stochastic Neighbor Embedding under f -divergences*. en. arXiv:1811.01247 [cs]. Nov. 2018. DOI: 10.48550/arXiv.1811.01247. URL: <http://arxiv.org/abs/1811.01247> (visited on 02/24/2026).

- [22] Anna Klimovskaia et al. “Poincaré maps for analyzing complex hierarchies in single-cell data”. en. In: *Nature Communications* 11.1 (June 2020), p. 2966. ISSN: 2041-1723. DOI: 10.1038/s41467-020-16822-4. URL: <https://www.nature.com/articles/s41467-020-16822-4> (visited on 11/21/2025).
- [23] Dmitry Kobak and George C. Linderman. “Initialization is critical for preserving global data structure in both t-SNE and UMAP”. en. In: *Nature Biotechnology* 39.2 (Feb. 2021), pp. 156–157. ISSN: 1087-0156, 1546-1696. DOI: 10.1038/s41587-020-00809-z. URL: <https://www.nature.com/articles/s41587-020-00809-z> (visited on 02/04/2026).
- [24] Eryk Kopczyński, Dorota Celińska, and Marek Čtrnáct. “HyperRogue: Playing with Hyperbolic Geometry”. In: *Proceedings of Bridges 2017: Mathematics, Art, Music, Architecture, Education, Culture*. Ed. by David Swart, Carlo H. Séquin, and Kristóf Fenyvesi. Phoenix, Arizona: Tessellations Publishing, 2017, pp. 9–16. ISBN: 978-1-938664-22-9. URL: <http://archive.bridgesmathart.org/2017/bridges2017-9.html>.
- [25] J. B. Kruskal. “Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis”. en. In: *Psychometrika* 29.1 (Mar. 1964), pp. 1–27. ISSN: 0033-3123, 1860-0980. DOI: 10.1007/BF02289565. URL: https://www.cambridge.org/core/product/identifier/S003331230004206X/type/journal_article (visited on 02/24/2026).
- [26] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791. URL: <https://ieeexplore.ieee.org/document/726791/> (visited on 06/24/2026).
- [27] George C. Linderman et al. “Efficient Algorithms for t-distributed Stochastic Neighborhood Embedding”. en. In: *Nature Methods* 16.3 (Mar. 2019). arXiv:1712.09005 [cs], pp. 243–245. ISSN: 1548-7091, 1548-7105. DOI: 10.1038/s41592-018-0308-4. URL: <http://arxiv.org/abs/1712.09005> (visited on 04/22/2026).
- [28] Moritz von Looz, Henning Meyerhenke, and Roman Prutkin. “Generating Random Hyperbolic Graphs in Subquadratic Time”. In: Nov. 2015, pp. 467–478. ISBN: 978-3-662-48970-3. DOI: 10.1007/978-3-662-48971-0_40.
- [29] Laurens van der Maaten. “Accelerating t-SNE using Tree-Based Algorithms”. In: *Journal of Machine Learning Research* 15 (2014).
- [30] Laurens van der Maaten, Eric O Postma, H Jaap Van Den Herik, et al. “Dimensionality reduction: A comparative review”. In: *Journal of Machine Learning Research* 10.1 (2009), pp. 1–41.
- [31] Laurens J. P. van der Maaten and Geoffrey E. Hinton. “Visualizing high-dimensional data using t-SNE”. In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.
- [32] Laurens van der Maaten. *t-SNE (t-Distributed Stochastic Neighbor Embedding)*. 2008. URL: <https://lvdmaaten.github.io/tsne/>.

- [33] Arpit Narechania et al. “VitaLITy: Promoting Serendipitous Discovery of Academic Literature with Transformers & Visual Analytics”. en. In: *IEEE Transactions on Visualization and Computer Graphics* 28.1 (Jan. 2022). arXiv:2108.03366 [cs], pp. 486–496. ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: 10.1109/TVCG.2021.3114820. URL: <http://arxiv.org/abs/2108.03366> (visited on 02/27/2026).
- [34] Maximilian Nickel and Douwe Kiela. *Poincaré Embeddings for Learning Hierarchical Representations*. en. arXiv:1705.08039 [cs]. May 2017. DOI: 10.48550/arXiv.1705.08039. URL: <http://arxiv.org/abs/1705.08039> (visited on 02/10/2026).
- [35] F. Pedregosa et al. *sklearn.manifold.TSNE — scikit-learn 1.x documentation*. 2025. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>.
- [36] N. Pezzotti et al. “Hierarchical Stochastic Neighbor Embedding”. en. In: *Computer Graphics Forum* 35.3 (June 2016), pp. 21–30. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.12878. URL: <https://onlinelibrary.wiley.com/doi/10.1111/cgf.12878> (visited on 02/24/2026).
- [37] Nicola Pezzotti et al. “Approximated and User Steerable tSNE for Progressive Visual Analytics”. In: *IEEE Transactions on Visualization and Computer Graphics* 23.7 (July 2017), pp. 1739–1752. ISSN: 1941-0506. DOI: 10.1109/TVCG.2016.2570755. URL: <https://ieeexplore.ieee.org/document/7473883/> (visited on 07/19/2026).
- [38] Nicola Pezzotti et al. “GPGPU Linear Complexity t-SNE Optimization”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (Jan. 2020), pp. 1172–1181. ISSN: 1941-0506. DOI: 10.1109/TVCG.2019.2934307. URL: <https://ieeexplore.ieee.org/document/8811606/> (visited on 07/19/2026).
- [39] Mireya Plass et al. “Cell type atlas and lineage tree of a whole complex animal by single-cell transcriptomics”. en. In: *Science* 360.6391 (May 2018), eaaq1723. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aaq1723. URL: <https://www.science.org/doi/10.1126/science.aaq1723> (visited on 07/31/2026).
- [40] John G. Ratcliffe. *Foundations of Hyperbolic Manifolds*. en. Vol. 149. Graduate Texts in Mathematics. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-31596-2 978-3-030-31597-9. DOI: 10.1007/978-3-030-31597-9. URL: <http://link.springer.com/10.1007/978-3-030-31597-9> (visited on 07/22/2026).
- [41] Christopher De Sa et al. *Representation Tradeoffs for Hyperbolic Embeddings*. en. arXiv:1804.03329 [cs]. Apr. 2018. DOI: 10.48550/arXiv.1804.03329. URL: <http://arxiv.org/abs/1804.03329> (visited on 02/10/2026).
- [42] Sagnik Sarkar. *Example Images of the MNIST Handwritten Digit Classification dataset*. en. URL: https://www.researchgate.net/figure/Example-Images-of-the-MNIST-Handwritten-Digit-Classification-dataset_fig2_359449733 (visited on 03/04/2026).

- [43] Liam Schoneveld. *Figure A.8: (a) Visualization of the t-SNE result of handwritten digits...* en. URL: https://www.researchgate.net/figure/Figure-A8-a-Visualization-of-the-t-SNE-result-of-handwritten-digits-using-sunspot_fig34_365303250 (visited on 11/21/2025).
- [44] Martin Skrodzki et al. “Accelerating Hyperbolic t-SNE”. en. In: *IEEE Transactions on Visualization and Computer Graphics* 30.7 (July 2024), pp. 4403–4415. ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: 10.1109/TVCG.2024.3364841. URL: <https://ieeexplore.ieee.org/document/10432970/> (visited on 11/09/2025).
- [45] Gilbert Strang. *Linear Algebra and Learning from Data*. Wellesley, MA: Wellesley-Cambridge Press, 2019.
- [46] Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. “A Global Geometric Framework for Nonlinear Dimensionality Reduction”. en. In: *Science* 290.5500 (Dec. 2000), pp. 2319–2323. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.290.5500.2319. URL: <https://www.science.org/doi/10.1126/science.290.5500.2319> (visited on 06/24/2026).
- [47] Benjamin Wilson and Matthias Leimeister. *Gradient descent in hyperbolic space*. en. Aug. 2018. URL: <https://arxiv.org/abs/1805.08207>.
- [48] Haoran Xia. “Revisiting Hyperbolic t-SNE”. en. MA thesis. Delft University of Technology, 2025.
- [49] Yuansheng Zhou and Tatyana O. Sharpee. “Hyperbolic geometry of gene expression”. en. In: *iScience* 24.3 (Mar. 2021), p. 102225. ISSN: 25890042. DOI: 10.1016/j.isci.2021.102225. URL: <https://linkinghub.elsevier.com/retrieve/pii/S2589004221001930> (visited on 11/11/2025).

Appendix A

Hyperbolic t-SNE Gradient Derivations

A.1 t-SNE Gradient Derivation for a Generic Distance Function

This derivation will mainly follow the work of [48] which in turn follows [31].

Let $r_{ij} = d_{ij}^2$, where d_{ij} is an arbitrary distance function between embeddings y_i and y_j .

The cost function is defined as:

$$\mathcal{C} = \text{KL}(P \parallel Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Applying the chain rule, and introducing Kronecker deltas,

$$\begin{aligned} \frac{\partial \mathcal{C}}{\partial y_{i\alpha}} &= \sum_{k \neq l} \frac{\partial \mathcal{C}}{\partial r_{kl}} \frac{\partial r_{kl}}{\partial y_{i\alpha}} \\ &= \sum_{k \neq l} \frac{\partial \mathcal{C}}{\partial r_{kl}} \left(\frac{\partial r_{kl}}{\partial y_{i\alpha}} \delta_{ik} + \frac{\partial r_{kl}}{\partial y_{i\alpha}} \delta_{il} \right) \\ &= \sum_l \frac{\partial \mathcal{C}}{\partial r_{il}} \frac{\partial r_{il}}{\partial y_{i\alpha}} + \sum_k \frac{\partial \mathcal{C}}{\partial r_{ki}} \frac{\partial r_{ki}}{\partial y_{i\alpha}} \\ &= \sum_l \frac{\partial \mathcal{C}}{\partial r_{il}} \frac{\partial r_{il}}{\partial y_{i\alpha}} + \sum_k \frac{\partial \mathcal{C}}{\partial r_{ik}} \frac{\partial r_{ik}}{\partial y_{i\alpha}} \\ &= 2 \sum_j \frac{\partial \mathcal{C}}{\partial r_{ij}} \frac{\partial r_{ij}}{\partial y_{i\alpha}} \\ &= 2 \sum_j \frac{\partial \mathcal{C}}{\partial r_{ij}} \frac{\partial r_{ij}}{\partial d_{ij}} \frac{\partial d_{ij}}{\partial y_{i\alpha}} \\ &= 4 \sum_j \frac{\partial \mathcal{C}}{\partial r_{ij}} d_{ij} \frac{\partial d_{ij}}{\partial y_{i\alpha}}. \end{aligned}$$

The partial derivatives w.r.t the squared distance can be split as follows:

$$\begin{aligned}
\frac{\partial \mathcal{C}}{\partial r_{ij}} &= - \sum_{k \neq l} p_{kl} \frac{\partial}{\partial r_{ij}} \log q_{kl} \\
&= - \sum_{k \neq l} p_{kl} \frac{\partial}{\partial r_{ij}} \log \left(\frac{(1 + r_{kl})^{-1}}{Z} \right) \\
&= - \left[\sum_{k \neq l} p_{kl} \frac{\partial}{\partial r_{ij}} \log(1 + r_{kl})^{-1} - \sum_{k \neq l} p_{kl} \frac{\partial}{\partial r_{ij}} \log Z \right].
\end{aligned}$$

Simplifying the first term,

$$\begin{aligned}
\sum_{k \neq l} p_{kl} \frac{\partial}{\partial r_{ij}} \log(1 + r_{kl})^{-1} &= - \sum_{k \neq l} p_{kl} (1 + r_{kl})^{-1} \delta_{ki} \delta_{lj} \\
&= -p_{ij} (1 + r_{ij})^{-1}.
\end{aligned}$$

Simplifying the second term using the fact that $\sum p_{kl} = 1$,

$$\begin{aligned}
\sum_{k \neq l} p_{kl} \frac{\partial}{\partial r_{ij}} \log Z &= \frac{\partial}{\partial r_{ij}} \log Z \\
&= \frac{1}{Z} \sum_{k \neq l} \frac{\partial}{\partial r_{ij}} (1 + r_{kl})^{-1} \\
&= -\frac{1}{Z} (1 + r_{ij})^{-2} \\
&= -q_{ij} (1 + r_{ij})^{-1}.
\end{aligned}$$

Combining the two terms, we arrive at a simple expression for the partial derivative with respect to the squared distance:

$$\begin{aligned}
\frac{\partial \mathcal{C}}{\partial r_{ij}} &= - [-p_{ij} (1 + r_{ij})^{-1} - q_{ij} (1 + r_{ij})^{-1}] \\
&= (p_{ij} - q_{ij}) (1 + r_{ij})^{-1}.
\end{aligned}$$

Substituting in our expression, we have a simple general form of the partial derivatives of the cost function:

$$\frac{\partial \mathcal{C}}{\partial y_{i\alpha}} = 4 \sum_j (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1} d_{ij} \frac{\partial d_{ij}}{\partial y_{i\alpha}}.$$

These partial derivatives do not themselves constitute the gradient in a curved space; rather, they serve as the building blocks from which the true gradient is constructed. Specifically, we first assemble them into the Euclidean gradient [1]:

$$\nabla_{y_i}^{\text{Euc}} C = \begin{pmatrix} \frac{\partial C}{\partial y_{i1}} \\ \vdots \\ \frac{\partial C}{\partial y_{id}} \end{pmatrix},$$

Note that "Euclidean" here refers to the metric used to compute the gradient's components, not to a claim that the embedding space itself is flat. The space can still be curved, and this curvature is accounted for separately through the metric tensor $G(y_i)$. We can use this Euclidean gradient together with the metric to yield the true gradient [1]:

$$\text{grad}_{y_i} C = G(y_i)^{-1} \nabla_{y_i}^{\text{Euc}} C$$

Now we will look at specific choices of distance function d_{ij} for different embedding spaces (note that q_{ij} is a function of d_{ij} and will change when we change embedding spaces).

A.1.1 Euclidean case

In the Euclidean case, the metric $G(y_i) = I$, so the Euclidean gradient is the true gradient. The Euclidean distance and its gradient are given by,

$$d_{ij} = \|y_i - y_j\|,$$

$$\frac{\partial d_{ij}}{\partial y_{i\alpha}} = \frac{y_{i\alpha} - y_{j\alpha}}{d_{ij}}.$$

As such, the gradient is:

$$\begin{aligned} \frac{\partial \mathcal{C}}{\partial y_{i\alpha}} &= 4 \sum_j (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1} d_{ij} \frac{\partial d_{ij}}{\partial y_{i\alpha}} \\ &= \boxed{4 \sum_j (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1} (y_{i\alpha} - y_{j\alpha})}. \end{aligned}$$

A.1.2 Poincaré disk

On the Poincaré disk, the metric is given by $G(y_i) = (\frac{2}{1-\|y_i\|^2})^2 I$ [44]. The distance metric and its gradient on the Poincaré disk are given as:

$$d_{ij} = \text{arccosh}(\gamma)$$

$$\frac{\partial d_{ij}}{\partial y_{i\alpha}} = \frac{4}{\beta_i^2 \beta_j \sqrt{\gamma^2 - 1}} [(\|y_i - y_j\|^2 + \beta_i) y_{i\alpha} - \beta_i y_{j\alpha}]$$

where $\beta_i = 1 - \|y_i\|^2$ is the distance of y_i from the edge of the disk, and $\gamma = 1 + 2 \frac{\|y_i - y_j\|^2}{\beta_i \beta_j}$. We verified that this form of the partial derivatives of the distance is equivalent to the form found in [44]. The gradient is given as,

$$\begin{aligned} [\text{grad}_{y_i} C]^\alpha &= [G(y_i)^{-1} \nabla_{y_i}^{\text{Euc}} C]^\alpha \\ &= \left(\frac{\beta_i^2}{4}\right) 4 \sum_j (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1} d_{ij} \frac{\partial d_{ij}}{\partial y_{i\alpha}} \\ &= \left(\frac{\beta_i^2}{4}\right) 4 \sum_j (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1} d_{ij} \frac{4}{\beta_i^2 \beta_j \sqrt{\gamma^2 - 1}} [(\|y_i - y_j\|^2 + \beta_i) y_{i\alpha} - \beta_i y_{j\alpha}] \\ &= \boxed{4 \sum_j (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1} d_{ij} \frac{1}{\beta_j \sqrt{\gamma^2 - 1}} [(\|y_i - y_j\|^2 + \beta_i) y_{i\alpha} - \beta_i y_{j\alpha}]} \end{aligned}$$

A.1.3 Hyperboloid model

Under the hyperboloid model, instead of having the embeddings be in $\mathbb{R}^n$, we instead use embeddings in $\mathbb{R}^{n+1}$. The metric is given by $dx_1^2 + \dots + dx_n^2 - dx_{n+1}^2$, which amounts to flipping the sign of the last component [8]. The inner product is,

$$\langle u, v \rangle_{n:1} = -u_{n+1}v_{n+1} + \sum_{i=1}^n u_i v_i.$$

The distance and its partial derivatives are given as [47],

$$\begin{aligned} d_{ij} &= \operatorname{arccosh}(-\langle \Theta_i, \Theta_j \rangle_{n:1}) \\ \frac{\partial d_{ij}}{\partial \Theta_{i\alpha}} &= -(\langle \Theta_i, \Theta_j \rangle_{n:1}^2 - 1)^{-1/2} \Theta_{j\alpha} (-1)^{\delta_{\alpha, n+1}} \end{aligned}$$

The gradient of the cost function in the ambient $\mathbb{R}^{n+1}$ space under the hyperboloid model is given as,

$$\begin{aligned} [\nabla_{\Theta_i}^{\mathbb{R}^{n+1}} C]_{\alpha} &= [G(\Theta_i)^{-1} \nabla_{\Theta_i}^{\text{Euc}} C]_{\alpha} \\ &= 4(-1)^{\delta_{\alpha, n+1}} \sum_j (p_{ij} - q_{ij})(1 + d_{ij}^2)^{-1} d_{ij} \frac{\partial d_{ij}}{\partial \Theta_{i\alpha}} \\ &= -4(-1)^{\delta_{\alpha, n+1}} \sum_j (p_{ij} - q_{ij})(1 + d_{ij}^2)^{-1} d_{ij} (\langle \Theta_i, \Theta_j \rangle_{n:1}^2 - 1)^{-1/2} \Theta_{j\alpha} (-1)^{\delta_{\alpha, n+1}} \\ &= -4 \sum_j (p_{ij} - q_{ij})(1 + d_{ij}^2)^{-1} d_{ij} (\langle \Theta_i, \Theta_j \rangle_{n:1}^2 - 1)^{-1/2} \Theta_{j\alpha} \end{aligned}$$

For ease of notation let us define

$$z_{ij} = (p_{ij} - q_{ij})(1 + d_{ij}^2)^{-1} d_{ij} (\langle \Theta_i, \Theta_j \rangle_{n:1}^2 - 1)^{-1/2}$$

which allows us to write the ambient gradient simply as,

$$[\nabla_{\Theta_i}^{\mathbb{R}^{n+1}} C]_{\alpha} = -4 \sum_j z_{ij} \Theta_{j\alpha}$$

For gradient descent, we want the gradient on the tangent space of the hyperboloid instead of in the ambient space. We compute the tangent space gradient by projecting the ambient space gradient on the tangent space as follows [47]:

$$[\nabla_{\Theta_i}^{\mathbb{H}^n} C]_{\alpha} = [\nabla_{\Theta_i}^{\mathbb{R}^{n+1}} C]_{\alpha} + \left\langle \Theta_i, \nabla_{\Theta_i}^{\mathbb{R}^{n+1}} C \right\rangle_{n:1} \Theta_{i\alpha}$$

Let us first find $\langle \Theta_i, \nabla_{\Theta_i}^{\mathbb{R}^{n+1}} C \rangle_{n:1}$:

$$\begin{aligned} \langle \Theta_i, \nabla_{\Theta_i}^{\mathbb{R}^{n+1}} C \rangle_{n:1} &= \langle \Theta_i, -4 \sum_j z_{ij} \Theta_j \rangle_{n:1} \\ &= -4 \sum_j z_{ij} \langle \Theta_i, \Theta_j \rangle_{n:1} \end{aligned}$$

Now we can find the tangent space gradient,

$$\begin{aligned}
[\nabla_{\Theta_i}^{\mathbb{H}^n} C]_\alpha &= [\nabla_{\Theta_i}^{\mathbb{R}^{n+1}} C]_\alpha + \left\langle \Theta_i, \nabla_{\Theta_i}^{\mathbb{R}^{n+1}} C \right\rangle_{n:1} \Theta_{i\alpha} \\
&= -4 \sum_j z_{ij} \Theta_{j\alpha} + (-4) \sum_j z_{ij} \langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_{i\alpha} \\
&= -4 \sum_j z_{ij} [\langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_{i\alpha} + \Theta_{j\alpha}] \\
&= \boxed{-4 \sum_j (p_{ij} - q_{ij})(1 + d_{ij}^2)^{-1} d_{ij} (\langle \Theta_i, \Theta_j \rangle_{n:1}^2 - 1)^{-1/2} [\langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_{i\alpha} + \Theta_{j\alpha}]}
\end{aligned}$$

The exponential map in the hyperboloid model is given by [47]:

$$\text{Exp}_\Theta(\xi) = \cosh(\|\xi\|_{n:1}) \Theta + \sinh(\|\xi\|_{n:1}) \frac{\xi}{\|\xi\|_{n:1}}.$$

For visualizing our points in the hyperboloid model, we can transform the points to points in the Poincaré model using the mapping,

$$y = \frac{1}{\Theta_{n+1} + 1}(\Theta_1, \dots, \Theta_n)$$

A.2 Proving the hyperboloid model gradient corresponds to the Poincaré model gradient via the differential map

In this section we will verify that the gradient in the hyperboloid model and the gradient in the Poincaré model are indeed equivalent under the differential map.

The differential $d\rho|_{\Theta_i} : T_{\Theta_i} \mathbb{H}^n \rightarrow T_{y_i} \mathbb{B}^n$ is used to convert the tangent space gradient in the hyperboloid model to the tangent space gradient of the Poincaré model [47]:

$$[d\rho|_{\Theta_i}(\nabla_{\Theta_i}^{\mathbb{H}^n} C)]_\alpha = \frac{1}{\Theta_{i,n+1} + 1} ([\nabla_{\Theta_i}^{\mathbb{H}^n} C]_\alpha - \frac{\Theta_{i\alpha} [\nabla_{\Theta_i}^{\mathbb{H}^n} C]_{n+1}}{\Theta_{i,n+1} + 1})$$

We can use the mapping from the Poincaré disk to the hyperboloid as follows [47]:

$$\begin{aligned}
\Theta_i &= \frac{2}{1 - \|y_i\|^2} (y_{i1}, \dots, y_{in}, \frac{1 + \|y_i\|^2}{2}) \\
&= \frac{2}{\beta_i} (y_{i1}, \dots, y_{in}, \frac{1 + \|y_i\|^2}{2})
\end{aligned}$$

Lemma 1. $-\langle \Theta_i, \Theta_j \rangle_{n:1} = \gamma$

Proof.

$$\begin{aligned}
 -\langle \Theta_i, \Theta_j \rangle_{n:1} &= \Theta_{i,n+1} \Theta_{j,n+1} - \sum_{\alpha=1}^n \Theta_{i_\alpha} \Theta_{j_\alpha} \\
 &= \frac{4}{\beta_i \beta_j} \left[\frac{(1 + \|y_i\|^2)(1 + \|y_j\|^2)}{4} - (y_i \cdot y_j) \right] \\
 &= \frac{2}{\beta_i \beta_j} \left[\frac{(1 + \|y_i\|^2)(1 + \|y_j\|^2)}{2} - 2(y_i \cdot y_j) \right]
 \end{aligned}$$

$$\begin{aligned}
 (1 + \|y_i\|^2)(1 + \|y_j\|^2) &= \|y_i\|^2 \|y_j\|^2 + \|y_i\|^2 + \|y_j\|^2 + 1 \\
 &= (1 - \beta_i)(1 - \beta_j) + \|y_i - y_j\|^2 + 2(y_i \cdot y_j) + 1 \\
 &= \beta_i \beta_j - \beta_i - \beta_j + 1 + \|y_i - y_j\|^2 + 2(y_i \cdot y_j) + 1 \\
 &= \beta_i \beta_j + \|y_i\|^2 + \|y_j\|^2 + \|y_i - y_j\|^2 + 2(y_i \cdot y_j) \\
 &= \beta_i \beta_j + 2\|y_i - y_j\|^2 + 4(y_i \cdot y_j)
 \end{aligned}$$

$$\begin{aligned}
 \frac{2}{\beta_i \beta_j} \left[\frac{(1 + \|y_i\|^2)(1 + \|y_j\|^2)}{2} - 2(y_i \cdot y_j) \right] &= \frac{2}{\beta_i \beta_j} \left[\frac{\beta_i \beta_j + 2\|y_i - y_j\|^2 + 4(y_i \cdot y_j)}{2} - 2(y_i \cdot y_j) \right] \\
 &= 1 + \frac{2}{\beta_i \beta_j} [\|y_i - y_j\|^2 + 2(y_i \cdot y_j) - 2(y_i \cdot y_j)] \\
 &= 1 + \frac{2}{\beta_i \beta_j} \|y_i - y_j\|^2 \\
 &= \gamma
 \end{aligned}$$

□

Lemma 2.

$$\frac{1}{\Theta_{i,n+1} + 1} = \frac{\beta_i}{2}$$

Proof.

$$\begin{aligned}
 \frac{1}{\Theta_{i,n+1} + 1} &= \frac{1}{\frac{1 + \|y_i\|^2}{\beta_i} + 1} \\
 &= \frac{\beta_i}{1 + \|y_i\|^2 + \beta_i} \\
 &= \frac{\beta_i}{(2 - \beta_i) + \beta_i} \\
 &= \frac{\beta_i}{2}
 \end{aligned}$$

□

Theorem 1. *The pushforward of the hyperboloid gradient is indeed the Poincaré gradient:*

$$d\rho|_{\Theta_i}(\nabla_{\Theta_i}^{\mathbb{H}^n} C) = \nabla_{y_i}^{\mathbb{B}^n} C$$

Proof. First we can note that both hyperboloid and Poincaré gradients can be written using a common factor z_{ij} :

$$z_{ij} = -4(p_{ij} - q_{ij}) \frac{d_{ij}}{1 + d_{ij}^2} \frac{1}{\sqrt{\gamma^2 - 1}}$$

The hyperboloid gradient can be written as,

$$[\nabla_{\Theta_i}^{\mathbb{H}^n} C]_\alpha = \sum_j z_{ij} [\langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_{i\alpha} + \Theta_{j\alpha}]$$

while the Poincaré gradient can be written as,

$$[\nabla_{y_i}^{\mathbb{B}^n} C]_\alpha = - \sum_j z_{ij} \frac{1}{\beta_j} [(\|y_i - y_j\|^2 + \beta_i) y_{i\alpha} - \beta_i y_{j\alpha}].$$

Applying the differential to the hyperboloid gradient,

$$\begin{aligned} [d\rho|_{\Theta_i}(\nabla_{\Theta_i}^{\mathbb{H}^n} C)]_\alpha &= \frac{1}{\Theta_{i,n+1} + 1} ([\nabla_{\Theta_i}^{\mathbb{H}^n} C]_\alpha - \frac{\Theta_{i\alpha} [\nabla_{\Theta_i}^{\mathbb{H}^n} C]_{n+1}}{\Theta_{i,n+1} + 1}) \\ &= \frac{\beta_i}{2} ([\nabla_{\Theta_i}^{\mathbb{H}^n} C]_\alpha - \frac{\beta_i}{2} \Theta_{i\alpha} [\nabla_{\Theta_i}^{\mathbb{H}^n} C]_{n+1}) \\ &= \frac{\beta_i}{2} \left[\sum_j z_{ij} (-\gamma \Theta_{i\alpha} + \Theta_{j\alpha}) + \frac{\beta_i}{2} \Theta_{i\alpha} (-\gamma \Theta_{i,n+1} + \Theta_{j,n+1}) \right] \\ &= \sum_j z_{ij} \left(-\gamma y_{i\alpha} + \frac{\beta_i y_{j\alpha}}{\beta_j} \right) + y_{i\alpha} - y_{i\alpha} \sum_j z_{ij} \left(-\gamma \frac{(1 + \|y_i\|^2)}{2} + \frac{\beta_i}{\beta_j} \frac{1 + \|y_j\|^2}{2} \right) \\ &= \sum_j z_{ij} \left[y_{i\alpha} \left(-\gamma + \gamma \frac{1 + \|y_i\|^2}{2} - \frac{\beta_i}{\beta_j} \frac{1 + \|y_j\|^2}{2} \right) + y_{i\alpha} \frac{\beta_i}{\beta_j} \right] \\ &= \sum_j \frac{z_{ij}}{\beta_j} \left[y_{i\alpha} \left(-\beta_j \gamma + \beta_j \gamma \frac{1 + \|y_i\|^2}{2} - \beta_i \frac{1 + \|y_j\|^2}{2} \right) + y_{i\alpha} \beta_i \right] \end{aligned}$$

Simplifying the inner expression,

$$\begin{aligned} -\beta_j \gamma + \beta_j \gamma \frac{1 + \|y_i\|^2}{2} - \beta_i \frac{1 + \|y_j\|^2}{2} &= (\beta_j \gamma) \left(\frac{1 + \|y_i\|^2}{2} - 1 \right) - \beta_i \frac{1 + \|y_j\|^2}{2} \\ &= \left(\beta_j + \frac{2\|y_i - y_j\|^2}{\beta_i} \right) \left(-\frac{\beta_i}{2} \right) - \beta_i \frac{1 + \|y_j\|^2}{2} \\ &= -\frac{\beta_i \beta_j}{2} - \|y_i - y_j\|^2 - \beta_i \frac{1 + \|y_j\|^2}{2} \\ &= -\frac{\beta_i \beta_j}{2} - \|y_i - y_j\|^2 - \beta_i \frac{2 - \beta_j}{2} \\ &= -\|y_i - y_j\|^2 - \beta_i \\ &= -(\|y_i - y_j\|^2 + \beta_i) \end{aligned}$$

Substituting our simplified expression in the differential of the hyperboloid gradient, we arrive at the Poincaré gradient:

$$\begin{aligned} \sum_j \frac{z_{ij}}{\beta_j} \left[y_{i\alpha} (-\beta_j \gamma + \beta_j \gamma \frac{1 + \|y_i\|^2}{2} - \beta_i \frac{1 + \|y_j\|^2}{2}) + y_{j\alpha} \beta_i \right] &= - \sum_j \frac{z_{ij}}{\beta_j} [y_{i\alpha} (\|y_i - y_j\|^2 + \beta_i) - y_{j\alpha} \beta_i] \\ &= [\nabla_{y_i}^{\mathbb{B}^n} C]_\alpha \end{aligned}$$

□

Appendix B

HIt-SNE Derivations

B.1 HIt-SNE Kernel and Source Derivation

Here we show that the repulsive force $F_{\text{rep},i}$ can be written as a sum of kernel and source functions in a compact form as described in Section 3.1.3.

$$\begin{aligned}
F_{\text{rep},i} &= \sum_j q_{ij} (1 + d_{ij}^2)^{-1} d_{ij} (\langle \Theta_i, \Theta_j \rangle_{n:1}^2 - 1)^{-1/2} [\langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_i + \Theta_j] \\
&= \frac{1}{Z} \sum_j (1 + d_{ij}^2)^{-2} d_{ij} (\langle \Theta_i, \Theta_j \rangle_{n:1}^2 - 1)^{-1/2} [\langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_i + \Theta_j] \\
&= \frac{1}{Z} \sum_j K_2(\Theta_i, \Theta_j) [\langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_i + \Theta_j] \\
&= \frac{1}{Z} \sum_j K_2(\Theta_i, \Theta_j) \langle \Theta_i, \Theta_j \rangle_{n:1} \Theta_i + \frac{1}{Z} \sum_j K_2(\Theta_i, \Theta_j) \Theta_j \\
&= \frac{1}{Z} \Theta_i \langle \Theta_i, \sum_j K_2(\Theta_i, \Theta_j) \Theta_j \rangle_{n:1} + \frac{1}{Z} \sum_j K_2(\Theta_i, \Theta_j) \Theta_j \\
&= \frac{1}{Z} (\Theta_i \langle \Theta_i, \mathbf{X}_i \rangle_{n:1} + \mathbf{X}_i)
\end{aligned}$$

The normalizing constant can be computed as follows:

$$\begin{aligned}
Z &= \sum_i \sum_j (1 + d_{ij}^2)^{-1} \\
&= \sum_i \sum_j K_1(\Theta_i, \Theta_j) \cdot 1 \\
&= \sum_i \phi_{K_1, f_1}
\end{aligned}$$

B.2 Interpolation Error Bounds

In this section we will expand the interpolation error bound from the FIt-SNE paper. First in Appendix B.2.1 we will expand the original 1D case as mentioned in the paper to 2D. Then in Appendix B.2.2 we will expand the 2D FIt-SNE error bound to HIt-SNE. Note that this error bound as found in the FIt-SNE is not a bound on the total error, but instead a bound on the error from the interactions within a tile.

B.2.1 Bound on the interpolation error (FIt-SNE)

The following follows the proof in the one-dimensional case found in [27]. Let $\mathbf{y}_i^{(t)} \in \mathbb{R}^2$, for $i = 1, \dots, n_t$, denote the embeddings located in tile t , where n_t is the number of points in that tile. Let $\mathbf{a}^{(t)}$ and $\mathbf{b}^{(t)}$ be the bottom-left and top-right corners of tile t , respectively. The interpolation nodes in each direction are then chosen as

$$\xi_1^{(t)}, \dots, \xi_p^{(t)} \in [a_1^{(t)}, b_1^{(t)}]$$

along the x -coordinate, and

$$\eta_1^{(t)}, \dots, \eta_p^{(t)} \in [a_2^{(t)}, b_2^{(t)}]$$

along the y -coordinate. These are collected into the two-dimensional interpolation nodes $\gamma_{ab}^{(t)} = (\xi_a^{(t)}, \eta_b^{(t)})$ for $a, b = 1, \dots, p$.

The degree- $(p-1)$ 1D Lagrange basis in the x -coordinate is given by

$$\ell_a^{(t,1)}(x) = \prod_{\substack{r=1 \\ r \neq a}}^p \frac{x - \xi_r^{(t)}}{\xi_a^{(t)} - \xi_r^{(t)}}, \quad a = 1, \dots, p,$$

and the 1D Lagrange basis in the y -coordinate is similarly given by

$$\ell_b^{(t,2)}(y) = \prod_{\substack{r=1 \\ r \neq b}}^p \frac{y - \eta_r^{(t)}}{\eta_b^{(t)} - \eta_r^{(t)}}, \quad b = 1, \dots, p.$$

The degree- $(p-1)$ 2D Lagrange basis is the tensor product of these two 1D bases:

$$L_{ab}^{(t)}(\mathbf{y}) = \ell_a^{(t,1)}(y_1) \ell_b^{(t,2)}(y_2).$$

Let $K_p^{(t)}(\mathbf{y}, \mathbf{z})$ be the bivariate polynomial interpolant of the kernel $K(\mathbf{y}, \mathbf{z})$, whose values agree with those of K at the interpolation nodes:

$$K_p^{(t)}(\gamma_{ab}^{(t)}, \gamma_{cd}^{(t)}) = K(\gamma_{ab}^{(t)}, \gamma_{cd}^{(t)}).$$

We now show that $K_p^{(t)}(\mathbf{y}, \mathbf{z})$ can be written in terms of the Lagrange polynomials defined above:

$$\begin{aligned} K_p^{(t)}(\mathbf{y}, \mathbf{z}) &= \sum_{a,b} K(\gamma_{ab}^{(t)}, \mathbf{z}) L_{ab}^{(t)}(\mathbf{y}) \\ &= \sum_{a,b} \left(\sum_{c,d} K(\gamma_{ab}^{(t)}, \gamma_{cd}^{(t)}) L_{cd}^{(t)}(\mathbf{z}) \right) L_{ab}^{(t)}(\mathbf{y}) \\ &= \sum_{a,b} \sum_{c,d} K(\gamma_{ab}^{(t)}, \gamma_{cd}^{(t)}) L_{ab}^{(t)}(\mathbf{y}) L_{cd}^{(t)}(\mathbf{z}). \end{aligned}$$

Let $\tilde{\phi}^{(t)}(\mathbf{y}_i)$ denote the approximation to $\phi^{(t)}(\mathbf{y}_i)$, the contribution to $\phi(\mathbf{y}_i)$ from the points in tile t :

$$\phi^{(t)}(\mathbf{y}_i) = \sum_{j=1}^{n_t} K(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) f_j^{(t)}, \quad \tilde{\phi}^{(t)}(\mathbf{y}_i) = \sum_{j=1}^{n_t} K_p^{(t)}(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) f_j^{(t)},$$

where $f_j^{(t)}$ is the source value associated with the point $\mathbf{y}_j^{(t)}$.

To bound the kernel sum error $\left| \tilde{\phi}^{(t)}(\mathbf{y}_i) - \phi^{(t)}(\mathbf{y}_i) \right|$, we first assume a uniform bound on the interpolation error over the tile:

$$\sup_{\mathbf{y}, \mathbf{z} \in [a_1^{(t)}, b_1^{(t)}] \times [a_2^{(t)}, b_2^{(t)}]} \left| K_p^{(t)}(\mathbf{y}, \mathbf{z}) - K(\mathbf{y}, \mathbf{z}) \right| \leq \varepsilon.$$

The kernel sum error is then bounded as follows:

$$\begin{aligned} \left| \tilde{\phi}^{(t)}(\mathbf{y}_i) - \phi^{(t)}(\mathbf{y}_i) \right| &= \left| \sum_{j=1}^{n_t} \left(K_p^{(t)}(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) - K(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) \right) f_j^{(t)} \right| \\ &\leq \sum_{j=1}^{n_t} \left| K_p^{(t)}(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) - K(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) \right| \left| f_j^{(t)} \right| \\ &\leq \varepsilon \sum_{j=1}^{n_t} \left| f_j^{(t)} \right|. \end{aligned}$$

For the case of computing the normalization constant Z , the source function is $f = 1$ and,

$$\left| \tilde{\phi}_{K_1, f_1}^{(t)}(\mathbf{y}_i) - \phi_{K_1, f_1}^{(t)}(\mathbf{y}_i) \right| \leq \varepsilon n_t,$$

which shows that the error bound is proportional to the number of points in a the tile which can be reduced by refining the grid.

B.2.2 Bound on the interpolation error (HIt-SNE)

Let the map ℓ_t from embeddings in tile t to their local tangent coordinates be defined as

$$\ell_t(\Theta) = \log_{\mathbf{0}}(\rho(\Lambda_t^{-1}(\Theta))),$$

so that the coordinates of the embeddings in the local tangent space are given by

$$\mathbf{y}_i^{(t)} = \ell_t(\Theta_i^{(t)}), \quad i = 1, \dots, n_t.$$

The interpolation nodes $\mathbf{z}_j^{(t)} \in [a_1^{(t)}, b_1^{(t)}] \times [a_2^{(t)}, b_2^{(t)}]$ and the bounding box $\mathbf{a}^{(t)}, \mathbf{b}^{(t)}$ are as given in Section 3.2, where the corresponding nodes on the manifold are defined as $\Gamma_j^{(t)} = \ell_t^{-1}(\mathbf{z}_j^{(t)})$.

Define the pullback of the kernel to the tangent coordinates of tile t as

$$\kappa^{(t)}(\mathbf{y}, \mathbf{z}) = K(\ell_t^{-1}(\mathbf{y}), \ell_t^{-1}(\mathbf{z})), \quad \kappa^{(t)} : \left([a_1^{(t)}, b_1^{(t)}] \times [a_2^{(t)}, b_2^{(t)}] \right)^2 \rightarrow \mathbb{R},$$

so that in particular $\kappa^{(t)}(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) = K(\Theta_i^{(t)}, \Theta_j^{(t)})$. Let $\kappa_p^{(t)}$ be the degree- $(p-1)$ bivariate polynomial interpolant of $\kappa^{(t)}$, agreeing with it at the nodes $\mathbf{z}_j^{(t)}$, and assume the uniform bound

$$\sup_{\mathbf{y}, \mathbf{z} \in [a_1^{(t)}, b_1^{(t)}] \times [a_2^{(t)}, b_2^{(t)}]} \left| \kappa_p^{(t)}(\mathbf{y}, \mathbf{z}) - \kappa^{(t)}(\mathbf{y}, \mathbf{z}) \right| \leq \varepsilon.$$

The approximation to $\phi^{(t)}(\boldsymbol{\Theta}_i^{(t)})$ is then obtained by evaluating the interpolant in the tangent coordinates,

$$\tilde{\phi}^{(t)}(\boldsymbol{\Theta}_i^{(t)}) = \sum_{j=1}^{n_t} \kappa_p^{(t)}(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) f_j^{(t)},$$

and the error of the kernel sum is bounded by

$$\begin{aligned} \left| \tilde{\phi}^{(t)}(\boldsymbol{\Theta}_i^{(t)}) - \phi^{(t)}(\boldsymbol{\Theta}_i^{(t)}) \right| &= \left| \sum_{j=1}^{n_t} \left(\kappa_p^{(t)}(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) - \kappa^{(t)}(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) \right) f_j^{(t)} \right| \\ &\leq \sum_{j=1}^{n_t} \left| \kappa_p^{(t)}(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) - \kappa^{(t)}(\mathbf{y}_i^{(t)}, \mathbf{y}_j^{(t)}) \right| \left| f_j^{(t)} \right| \\ &\leq \varepsilon \sum_{j=1}^{n_t} \left| f_j^{(t)} \right|. \end{aligned}$$