The Numerical Stability of Hyperbolic Representation Learning

Gal Mishne [1] **Zhengchao Wan** [1] **Yusu Wang** [1] **Sheng Yang** [2]

Abstract

The hyperbolic space is widely used for representing hierarchical datasets due to its ability to embed trees with small distortion. However, this property comes at a price of numerical instability such that training hyperbolic learning models will sometimes lead to catastrophic NaN problems, encountering unrepresentable values in floating point arithmetic. In this work, we analyze the limitations of two popular models for the hyperbolic space, namely, the Poincaré ball and the Lorentz model. We find that, under the 64 bit arithmetic system, the Poincaré ball has a relatively larger capacity than the Lorentz model for correctly representing points. However, the Lorentz model is superior to the Poincaré ball from the perspective of optimization, which we theoretically validate. To address these limitations, we identify one Euclidean parametrization of the hyperbolic space which can alleviate these issues. We further extend this Euclidean parametrization to hyperbolic hyperplanes and demonstrate its effectiveness in improving the performance of hyperbolic SVM.

1. Introduction

The n-dimensional hyperbolic space $\mathbb{H}^n$ is the unique simply-connected Riemannian manifold with a constant sectional curvature -1. A remarkable property of $\mathbb{H}^n$ against the Euclidean space $\mathbb{R}^n$ is that the volume of a ball in $\mathbb{H}^n$ grows exponentially with respect to the radius. One can thus embed finite trees into $\mathbb{H}^n$ with arbitrarily small distortion (Sarkar, 2011). This motivates the study of representation learning of hierarchical data into hyperbolic space (Nickel & Kiela, 2017) and, moreover, the design of deep neural networks in hyperbolic spaces (Ganea et al., 2018; Chen et al., 2022), with applications in various domains where hierarchical data is abundant, such as NLP (Zhu et al., 2020; López et al., 2019; López & Strube, 2020), recommendation systems (Chamberlain et al., 2019) and neuroscience (Gao et al., 2020).

However, the exponential volume growth property leads to numerical instability in training hyperbolic learning models. As shown by (Sala et al., 2018), in order to represent points in the popular Poincaré model (Nickel & Kiela, 2017; Ganea et al., 2018), one requires a large number of bits to avoid undesirable rounding errors when dealing with small numbers. The Lorentz model, a popular alternative for representing the hyperbolic space (Nickel & Kiela, 2018; Law et al., 2019), suffers an opposite numerical issue in dealing with large numbers. (Yu & De Sa, 2019) proved that representing Lorentz points using floating number arithmetic could lead to a huge representation error when the points are far from the origin. Besides, in terms of optimization and training, the empirical superiority of the Lorentz model over the Poincaré model has been often observed in tasks such as word embeddings (Law et al., 2019; Nickel & Kiela, 2018), however, the reason behind is yet unclear.

It is thus important to theoretically clarify the practical limitations of current hyperbolic models and to explain why certain models have better empirical performance than others. The goal of this paper is to address these issues and to provide a new model with better numerical performance.

Our Contributions (1) We are the first to clearly identify the representation capacity of two popular hyperbolic models, the Lorentz and the Poincaré models, in a geometric lens under the setting of 64-bit system[1]. (2) We provide a theoretical study of optimization on hyperbolic spaces from the perspective of the gradient vanishing issue inherent to hyperbolic models. We confirm theoretically that the Lorentz model is better than the Poincaré model from this aspect, which was previously only an empirical observation. (3) We study a simple yet effective Euclidean parameterization of the hyperbolic space which turns out to alleviate the numerical issues present in the Lorentz and Poincaré models. Although this Euclidean parametrization has been

[1]Halıcıoğlu Data Science Institute, University of California San Diego, La Jolla, California, USA [2]Harvard John A. Paulson School of Engineering and Applied Science, Harvard University, Cambridge, Massachusetts, USA. Correspondence to: Zhengchao Wan <z5wan@ucsd.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1]Standard CPU/GPU processors have a max precision of 64 bits, and surpassing this limit requires ticklish custom implementations.

utilized in the literature (see for example (Mathieu et al., 2019)), we provide new insights into this construction by theoretically showing that the Euclidean parameterization is as good as the Lorentz model in terms of optimization; but it has no limitation in representation capacity. We also empirically validate that the Euclidean parameterization improves the performance in tree embedding tasks compared to both the Lorentz and Poincaré models. Moreover, we apply our Euclidean parametrization to hyperbolic SVM and propose the method LSVMPP, which has superior performance in comparison to other methods in our experiments.

2. Preliminary

In this section, we provide the necessary background for the n-dimensional hyperbolic space $\mathbb{H}^n$. There are multiple isometric models for describing $\mathbb{H}^n$ (see (Peng et al., 2021) for a detailed survey). In this paper, we mainly focus on the Poincaré ball and the Lorentz model.

Poincaré Ball The Poincaré ball is the unit open ball $\mathbb{D}^n \subseteq \mathbb{R}^n$ with a Riemannian metric conformal to the underlying Euclidean one: at each point $x \in \mathbb{D}^n$, the metric $g_x = \lambda_x^2 h_x$, where h_x stands for the Euclidean metric at x and $\lambda_x := \frac{2}{(1-\|x\|^2)}$. For any $x, y \in \mathbb{D}^n$, the geodesic distance between them is expressed as

$$d_{\mathbb{D}^n}(x,y) = \operatorname{arccosh}\left(1 + \frac{2\|x-y\|^2}{(1-\|x\|^2)(1-\|y\|^2)}\right) \quad (1)$$

Lorentz Model The Lorentz model interprets $\mathbb{H}^n$ as a submanifold of the so-called Minkowski space $\mathbb{R}^{n,1}$. This is the linear space $\mathbb{R}^{n+1}$ equipped with the Minkowski product $[x,y] := -x_0y_0 + \sum_{i=1}^n x_iy_i$. The Lorentz model (also called the hyperboloid) is then the n-dim "unit sphere" in the Minkowski space:

$$\mathbb{L}^n := \{x \in \mathbb{R}^{n,1} : x_0 > 0,\ [x,x] = -1\}.$$

For any $x, y \in \mathbb{L}^n$, the geodesic distance between them is

$$d_{\mathbb{L}^n}(x,y) = \operatorname{arccosh}(-[\mathrm{x},\mathrm{y}]) \quad (2)$$

Notation for Norms and Gradients Let $\|\cdot\|$ and $\langle\cdot,\cdot\rangle$ denote Euclidean l_2 norm and inner product. We use $\|\cdot\|_{\mathbb{D}^n}$ and $\|\cdot\|_{\mathbb{L}^n}$ to denote norms of Poincaré and Lorentz vectors. More precisely, for any $v \in T_x\mathbb{D}^n$ and $w \in T_y\mathbb{L}^n$,

$$\|v\|_{\mathbb{D}^n} = \lambda_x \|v\| \quad \text{and} \quad \|w\|_{\mathbb{L}^n} = \sqrt{[w,w]}.$$

Similarly, we use ∇ to denote Euclidean gradient and $\nabla_{\mathbb{D}^n}, \nabla_{\mathbb{L}^n}$ for Poincaré and Lorentz gradient.

Transition Between the Two Models The isometry between the two models can be written explicitly as $\varphi : \mathbb{D}^n \to \mathbb{L}^n$ which sends $x = (x_1, \ldots, x_n)$ to $y := \left(\frac{\|x\|^2+1}{1-\|x\|^2}, \frac{2x_1}{1-\|x\|^2}, \ldots, \frac{2x_n}{1-\|x\|^2}\right)$. Its inverse is $\psi : \mathbb{L}^n \to \mathbb{D}^n$ which sends $y = (y_0, \ldots, y_n)$ to $x := (\frac{y_1}{1+y_0}, \ldots, \frac{y_n}{1+y_0})$. The origin 0 in $\mathbb{D}^n$ corresponds to the point $\bar{0} = (1, 0, \ldots, 0)$ in $\mathbb{L}^n$ under these isometries and we call $\bar{0}$ the origin of the Lorentz model.

Operations on $\mathbb{H}^n$ In Riemannian geometry, there are several important operations relating tangent vectors and points on manifolds, such as the exponential map, the logarithm map, and parallel transport, which can be written explicitly when considering either model of $\mathbb{H}^n$.

For example, in the Poincaré ball, the exponential map for any $x \in \mathbb{D}^n$ and any $v \in T_x\mathbb{D}^n$ is given by

$$\exp_x(v) = x \oplus \tanh\left(\frac{\lambda_x}{2}\|v\|\right)\frac{v}{\|v\|}, \quad (3)$$

where $\oplus$ denotes the so-called Möbius addition (Ungar, 2001). In the Lorentz model, the exponential map for any $x \in \mathbb{L}^n$ and any $v \in T_x\mathbb{L}^n$ is given by

$$\exp_x(v) = \cosh(\|v\|_{\mathbb{L}^n})x + \sinh(\|v\|_{\mathbb{L}^n})\frac{v}{\|v\|_{\mathbb{L}^n}}. \quad (4)$$

3. Comparing Lorentz and Poincaré Models

As mentioned in the introduction, the volume of a ball in $\mathbb{H}^n$ grows exponentially w.r.t. the radius. This results in that finite trees can be embedded into $\mathbb{H}^n$ with arbitrarily small distortion (Sarkar, 2011). However, in order to reach desired small distortion, one needs to scale the embedded tree by a large factor by pushing embedded points towards infinity in $\mathbb{H}^n$ as much as possible, and this comes at a price of numerical representation.

Poincaré Ball To embed a combinatorial tree T with diameter l and maximal degree D up to distortion $(1+\varepsilon)$, one needs $\Omega(\frac{l\log(D)}{\varepsilon})$ bits to correctly represent embedded points in the Poincaré ball to avoid these points being rounded off to the boundary (Sala et al., 2018).

We first interpret the number of bits in representing Poincaré points through a more geometrical lens.

Proposition 3.1 (Poincaré radius)**.** *For any point $x \in \mathbb{D}^n$, if $\|x\| = 1 - 10^{-k}$ for some positive number k, then in fact,*

$$d_{\mathbb{D}^n}(0,x) = \ln(10)k + \ln(2) + O(10^{-k}).$$

See Appendix A.1 for proof. Here k is the number of bits required to avoid x being rounded to the boundary. Under the Float64 arithmetic system, the rounding error for the

subtraction $1-10^{-k}$ is 2^{-53}, i.e., $1-10^{-k}$ will be rounded to 1 when $10^{-k} \leq 2^{-53} \approx 10^{-16}$. Hence, the maximum k is around 16. This corresponds to a distance $d_{\mathbb{D}^n}(0,x) \approx 38$. Thus, we can only represent points correctly within a ball of radius $r_0 \approx 38$ in the Poincaré ball.

Lorentz Model Interestingly, it turns out that the Lorentz model has an even smaller representation capacity: it can only represent points correctly within a ball of radius $r_0/2 \approx 19$. Recall the fundamental equation $[x,x] = -1$ in defining the Lorentz model. Given a point $x = (x_0,\ldots,x_n) \in \mathbb{L}^n$, if $x_0 = 10^8$, then $x_0^2 = 10^{16}$. Then, the floating number representation of $x_0^2 - 1$ is the same as the one for x_0^2 (roughly speaking, in the Float64 system, the sum/difference of two numbers differ in orders of magnitude over 16 would be rounded to the larger number itself), causing the computation of $[x,x]$ to return 0 instead of the desired -1. Moreover, due to the following result, we know that the condition $x_0 = 10^8$ corresponds to the radius $r_0/2$.

Proposition 3.2 (Lorentz radius)**.** *For any point $x \in \mathbb{L}^n$, if $x_0 = 10^k$ for some positive number k, then,*

$$d_{\mathbb{L}^n}(\bar{0},x) = \ln(10)k + \ln(2) + O(10^{-2k}).$$

See Appendix A.2 for proof. This difference in the radii between the two models can be clearly seen from the perspective of the exponential maps. We postpone this interpretation to later Remark 4.1.

The limitation of the Lorentz model is different from the Poincaré ball. In the Poincaré ball, the radius $r_0 \approx 38$ is a "hard" constraint: any point r_0 distant from the origin will collapse to the boundary and there is no way to consider any operations on the hyperbolic space such as computing the distance thereafter. However, in the Lorentz model, the radius $r_0/2 \approx 19$ is a "soft" constraint: when a point $x = (x_0, x_1, \ldots, x_n)$ is further than $r_0/2$ away from the origin, although there is no viable method to check whether the point is on the Lorentz model, we still have the coordinates to work with: if we let $x_r := (x_1,\ldots,x_n)$, although $x_0 = \sqrt{\|x_r\|^2 + 1}$ will be rounded off to $\|x_r\|$, there is no numerical constraint for the vector x_r. In this way, one can still cope with x_r and perform some operations on the Lorentz model. Together with the optimization superiority of the Lorentz model we mention in the next section, this may account for some empirical observations that the Lorentz model is more stable than the Poincaré ball.

We comment that many works either explicitly or implicitly impose different thresholds in their implementations to restrict all points within a certain radius ((Skopek et al., 2019), (Nickel & Kiela, 2017), and the popular manifold research toolbox package by (Kochurov et al., 2020), with limited discussion on the impact of the choice of these thresholds on the representation capacity and other performance. *The discussion in this section fills the gap and provides a guide for the choice of thresholds.*

Finally, while the concepts of representation capacity for the Poincaré ball and the Lorentz model initially appear to be distinct, they can be unified through the examination of exponential maps. Although the representation capacity is traditionally defined for points on a manifold within these two models, it can be equivalently specified in the tangent space. This is facilitated by the exponential map $\exp_x : T_x\mathbb{H}^n \to \mathbb{H}^n$, which preserves distance in the radial direction, i.e., $\|v\| = d_{\mathbb{H}^n}(x, \exp_x(v))$ for any $v \in T_x\mathbb{H}^n$.

We then further introduce the concept of *numerical representation capacity* for a hyperbolic model. This is formally defined as the radius of the largest ball, centered at the origin in the tangent space, with the property that all points within this ball can be accurately represented within the hyperbolic model via the exponential map. Now, under the exponential map formula Equation (3) for the Poincaré ball, if we take our reference point x to be the origin, then vectors with length larger than 38 will be mapped to the boundary of $\mathbb{D}^n$ which is outside of the manifold. Similarly, in the case of the Lorentz model, under the exponential map formula Equation (4), if we choose $x = \bar{0} = (1,0,\ldots,0)$, then vectors with length larger than 19 will be mapped outside of the Lorentz model. Specifically, these vectors will be mapped to the cone $x_0^2 = \sum_{i=1}^n x_i^2$. See also later Remark 4.1 for more details. This way of viewing things allows us to see how the concept of numerical representation capacity serves to unite our previous discussions on representation capacity for both the Poincaré and Lorentz models.

3.1. Optimization

Although we see that the Poincaré ball has a larger capacity in representing points than the Lorentz model, we show in this section that this advantage will be wiped out when considering optimization processes on hyperbolic space.

Generally speaking, for any manifold $\mathcal{M}$ and a smooth function $f : \mathcal{M} \to \mathbb{R}$, in order to solve a Riemannian optimization problem:

$$x^* = \mathrm{argmin}_{x\in\mathcal{M}} f(x),$$

one can apply Riemannian (stochastic) gradient descent (Bonnabel, 2013) with learning rate $\eta > 0$:

$$x_{t+1} = \exp_{x_t}(-\eta\nabla f(x_t)). \tag{5}$$

Theoretically speaking, due to the isometry between $\mathbb{D}^n$ and $\mathbb{L}^n$, solving optimization problems in $\mathbb{D}^n$ and $\mathbb{L}^n$ via Riemannian gradient descent should return the "same" result. To describe this result properly, throughout this section, we will consider a fixed differentiable function $f : \mathbb{D}^n \to \mathbb{R}$. We consider its corresponding function $g : \mathbb{L}^n \to \mathbb{R}$ on

the Lorentz model, i.e., $g = f \circ \psi$ where $\psi : \mathbb{L}^n \to \mathbb{D}^n$ is the isometry specified in Section 2. The functions f and g should be regarded as the same function defined on the hyperbolic space $\mathbb{H}^n$ under two different chart systems. We also illustrate the relationship between f and g through the following commutative diagram.

$$\begin{tikzcd} \mathbb{L}^n \arrow[r, shift left, "\psi"] \arrow[dr, "g"'] & \mathbb{D}^n \arrow[l, shift left, "\varphi"] \arrow[d, "f"] \\ & \mathbb{R} \end{tikzcd}$$

In the analysis below, we use x to represent points in $\mathbb{D}^n$ and let $y = \varphi(x) \in \mathbb{L}^n$ denote its corresponding point in the Lorentz model. Since φ and ψ are isometries, the following result holds trivially.

Proposition 3.3 (Gradient descent is the "same" for both models)**.** *For any learning rate $\eta > 0$, we have that*

$$\psi(\exp_y(-\eta \nabla_{\mathbb{L}^n} g(y))) = \exp_x(-\eta \nabla_{\mathbb{D}^n} f(x)).$$

Despite this theoretical equivalence, when implementing the Riemannian gradient descent algorithm, the Poincaré model is more prone to incur the *gradient vanishing problem* than the Lorentz model. We first see this through the following computation of the Euclidean norm of Riemannian gradients[2]. Such norm indicates the magnitude of the coordinates involved in representing the tangent vectors.

Lemma 3.4. *For any $x \in \mathbb{D}^n$, assume that $\|x\| = 1 - \delta$ for some small positive δ. Then, we have that*

$$\|\nabla_{\mathbb{D}^n} f(x)\| = \Omega\left(\delta^2 \|\nabla f(x)\|\right),$$
$$\|\nabla_{\mathbb{L}^n} g(y)\| = O\left(\|\nabla f(x)\|\right).$$

See proof from Appendix A.4 and note that in a special case where the vectors x and $\nabla f(x)$ are parallel, the big O for $\|\nabla_{\mathbb{L}^n} g(y)\|$ can be replaced by Ω. Notice that δ is small when x is close to the boundary of the Poincaré ball. As a consequence, optimizing through the Poincaré model results in dealing with numbers smaller in order of magnitude than through the Lorentz model. This suggests a more pronounced gradient vanishing problem for the Poincaré model when points are near the boundary and a potential accumulation of rounding error, e.g., when points are on their way to the boundary to achieve a lower distortion for the task of tree embedding.

To clearly derive the limitation of the Poincaré model in optimization, we consider Taylor expansions of one step of the Riemannian gradient descent (Equation (5)) for Poincaré ball and the Lorentz model as follows.

[2]Tangent vectors can be represented using the coordinates provided in either the Poincaré or the Lorentz model.

Theorem 3.5. *Let $\delta = 10^{-k}$ be a small positive number. Without loss of generality, consider the point $x = (1 - \delta, 0, \ldots, 0) \in \mathbb{D}^n$. Assume that $\nabla f(x) = (\partial_1 f(x), 0, \ldots, 0)$[3] and $\partial_1 f(x) < 0$. Let $y := \varphi(x) \in \mathbb{L}^n$. If we let $E := \|\nabla f(x)\|$, then*

$$\exp_x(-\eta \nabla_{\mathbb{D}^n} f(x)) = (1 - 10^{-k} + O(\eta E 10^{-2k}), 0, \ldots, 0),$$
$$\exp_y(-\eta \nabla_{\mathbb{L}^n} g(y)) =$$
$$10^k \Big(1 - \frac{1}{2} \cdot 10^{-k} + \eta E 10^{-k} + O(\eta E 10^{-2k}),$$
$$1 - \frac{1}{2} \cdot 10^{-k} + \eta E 10^{-k} + O(\eta E 10^{-2k}), 0, \ldots, 0\Big).$$

See Appendix A.5 for proof. To interpret the results, assume for simplicity that $\eta = 1$ and $E = \|\nabla f\| = O(1)$ is bounded. Then, at each step of Riemannian gradient descent, the Poincaré ball needs to compute the sum of two numbers (i.e., $1 - 10^{-k}$ and $O(\eta E 10^{-2k}) = O(10^{-2k})$) that differ by $2k$ in orders of magnitude whereas in the Lorentz case only differ by k (i.e., $1 - \frac{1}{2} \cdot 10^{-k}$ and $\eta E 10^{-k} + O(\eta E 10^{-2k}) = O(10^{-k})$). We emphasize that the terms $O(10^{-2k})$ and $O(10^{-k})$ do not represent error terms, but instead indicate the update in one step of Riemannian gradient descent. As a result, the Poincaré model has a smaller update term, which causes it to suffer from a more severe gradient vanishing issue. In particular, when k is chosen to be 8, the gradient term $O(10^{-2k}) = O(10^{-16})$ which is around the same order as the rounding error. Hence this term will be neglected and the Poincaré gradient descent will be stuck. However, for the Lorentz model, if one ignores the numerical representation issue, then from the optimization perspective, Lorentz model allows δ to be as small as 10^{-16} without encountering the severe gradient vanishing problem the Poincaré ball suffers from. Of course one could use a large learning rate $\eta = 10^8$ so that the Poincaré gradient numerical representation is similar to the one for the Lorentz. But such a large learning rate will induce instability in the training process. Note by Proposition 3.1, $\delta = 10^{-8}$ corresponds to a point 19 away from the origin. Hence, even though the Poincaré ball can represent points up to 38 away from the origin, a large part of the region cannot be utilized in optimization.

4. Euclidean Parametrization of Hyperbolic Space

In the previous section, we see that the Poincaré model and the Lorentz model have different pros/cons from two different perspectives: point-representation and optimization. It is thus natural to ask whether there is a way to represent

[3]We only consider this direction as it is the most relevant direction in understanding the phenomenon of pushing points towards the boundary of $\mathbb{D}^n$.

hyperbolic space that could prevail in both perspectives: allowing to represent points within a relatively big region like the Poincaré ball (i.e., of radius 38) or even larger, while in terms of optimization behaves like the Lorentz model.

In this section, we argue in Section 4.1 that using a Euclidean parametrization for the hyperbolic space can help address these limitations. Such Euclidean parameterization for hyperbolic space has previously been utilized in (Mathieu et al., 2019) for training hyperbolic features involved in hyperbolic VAE. However, here we present a new perspective that can provide a numerically more robust proxy for optimization on hyperbolic space. In Section 4.2, we consider a polar coordinate form of this parameterization and extend it to parametrize hyperbolic hyperplanes through a more succinct derivation than the one given in (Shimizu et al., 2020) and we also correct a mistake in their derivation (see Remark 4.1 and Remark 4.4). Finally, we observe that the hyperplane parametrization resolves the nonconvexity condition for the Lorentz hyperplane. In this way, we further apply this parametrization to and show a performance boost of the Lorentz SVM (Cho et al., 2019).

4.1. Feature Parametrization

One property inherent to the negative curvature of the hyperbolic space is that the exponential map at any point is a diffeomorphism (see the Cartan–Hadamard theorem in Riemannian geometry). This gives rise to a natural Euclidean parametrization of features in the hyperbolic space via the tangent space and the exponential map: pick a point $p \in \mathbb{H}^n$ and then consider the exponential map $\exp_p : T_p\mathbb{H}^n \to \mathbb{H}^n$; as $T_p\mathbb{H}^n$ can be identified (see the section below) as $\mathbb{R}^n$, we then have a Euclidean parametrization of $\mathbb{H}^n$.

In either the Poincaré ball or the Lorentz model, there exists a canonical choice of point p: the origin 0 in Poincaré ball and the origin $\bar{0} := (1, 0, \dots, 0)$ in the Lorentz model. Consider the map $\mathbb{R}^n \to T_0\mathbb{D}^n$ sending z to $\frac{z}{2}$ and the map $\mathbb{R}^n \to T_{\bar{0}}\mathbb{L}^n$ sending z to $(0, z)$. These two maps are both isometries, i.e., $\|z\|$ agrees with the Riemannian norms of the images of z in $T_0\mathbb{D}^n$ and $T_{\bar{0}}\mathbb{L}^n$.

Now, we specify how hyperbolic features are recovered through respective exponential maps: let $z \in \mathbb{R}^n$, then z parametrizes Poincaré features

$$x = \exp_0\left(\frac{z}{2}\right) = \tanh\left(\frac{\|z\|}{2}\right)\frac{z}{\|z\|} \tag{6}$$

and Lorentz features

$$y = \exp_{\bar{0}}((0, z)) = \left(\cosh(\|z\|), \sinh(\|z\|)\frac{z}{\|z\|}\right). \tag{7}$$

We denote these maps by $F_D : \mathbb{R}^n \to \mathbb{D}^n$ and $F_L : \mathbb{R}^n \to \mathbb{L}^n$, respectively. They are, of course, sending a point $z \in \mathbb{R}^n$ to the "same" point in the two models under the isometry $\varphi : \mathbb{D}^n \to \mathbb{L}^n$, i.e., the following diagram commutes.

$$\begin{tikzcd} \mathbb{R}^n \arrow[r, "F_D"] \arrow[dr, "F_L"'] & \mathbb{D}^n \arrow[d, "\varphi"] \\ & \mathbb{L}^n \end{tikzcd}$$

Furthermore, these two maps preserve distances between a point and the origin, i.e.,

$$d_{\mathbb{D}^n}(0, x) = d_{\mathbb{L}^n}(\bar{0}, y) = \|z\|.$$

Remark 4.1 (The mysterious "2"). Note that the division by 2 in the Poincaré case (see Equation (6)) is crucial to make the above equation hold. The missing of this division by 2 is prevalent in the literature (Shimizu et al., 2020) and may raise an unfair comparison between Poincaré and Lorentz models (see more details in later Remark 4.4). We also remark that under the 64 bit arithmetic system, $\tanh(t)$ will be rounded to 1 as long as $t > 19$. Hence, one can compute $\exp_0$ correctly up to radius $r_0 \approx 2 * 19$. As for the Lorentz model, since the float representations of $\cosh(t)$ and $\sinh(t)$ will be the same as $t > 19$ (note that $\tanh(t) = \frac{\sinh(t)}{\cosh(t)}$), the Lorentz $\exp_{\bar{0}}$ can only represent points correctly within a ball with radius up to $r_0/2 \approx 19$.

This Euclidean parametrization of course allows *representing any point in the hyperbolic space without concern regarding numerical limitations* found in the case of the Poincaré ball or the Lorentz model. However, it is important to note that the Euclidean parametrization does not preserve the Riemannian structure of the hyperbolic space. Nonetheless, we consider this parametrization a viable way to train functions defined on hyperbolic spaces. Next we will provide a more precise analysis of the behavior of this parametrization in optimization.

Optimization Given this Euclidean parametrization, one can translate hyperbolic optimization problems into a Euclidean one: Given any differentiable function $f : \mathbb{D}^n \to \mathbb{R}$, we define $h : \mathbb{R}^n \to \mathbb{R}$ by letting $h := f \circ F_D$. Of course, if we consider the function $g := f \circ \psi : \mathbb{L}^n \to \mathbb{R}$ defined on the Lorentz model, the composition $g \circ F_L$ agrees with h; see the commutative diagram below.

$$\begin{tikzcd} \mathbb{R}^n \arrow[r, "F_D"] \arrow[dr, "F_L"'] \arrow[drr, bend right, "h"'] & \mathbb{D}^n \arrow[dr, "f"] & \\ & \mathbb{L}^n \arrow[u, "\psi"'] \arrow[r, "g"] & \mathbb{R} \end{tikzcd}$$

Hence, in the discussion below, for simplicity, we start with a function defined on the Poincaré ball.

Consider the optimization problem on the hyperbolic space $\mathbb{D}^n$ given by:

$$x^* = \operatorname{argmin}_{x \in \mathbb{D}^n} f(x). \tag{8}$$

This problem can be transformed into an optimization problem on the Euclidean space $\mathbb{R}^n$ through the Euclidean parametrization F_D, resulting in the following optimization problem:

$$z^* = \operatorname{argmin}_{z\in\mathbb{R}^n} h(z) = \operatorname{argmin}_{z\in\mathbb{R}^n} f(F_D(z)). \quad (9)$$

Since F_D is a bijective diffeomorphism, any optimizer z^* of Equation (9) corresponds to an optimizer $F_D(z^*)$ of Equation (8), and vice versa.

The newly obtained optimization problem will suffer less from the gradient vanishing problem than the Poincaré model and has a similar performance to the Lorentz gradient descent. We analyze the one step gradient descent w.r.t. h in a manner similar to Theorem 3.5 below.

Theorem 4.2. *Under the same assumptions and notation as in Theorem 3.5, if we let* $z := F_D^{-1}(x) = (2\operatorname{arctanh}(1-\delta), 0, \dots, 0) \in \mathbb{R}^n$, *then* $\|\nabla h(z)\| = \Omega(\delta \|\nabla f(x)\|)$ *and*

$$z - \eta\nabla h(z) = \ln(10)k + \ln(2) + (\eta\|\nabla f(x)\| - 1/2)10^{-k} + O(10^{-2k}).$$

See Appendix A.6 for proof. Similarly, as in the discussion below Theorem 3.5, we assume $\eta = 1$ and $\|\nabla f(x)\| = O(1)$. In this case, we see the gradient update step involves the sum of two terms whose orders of magnitude differ roughly by $\log_{10}(k) + k$. This difference in orders of magnitude sits between the Poincaré case and the Lorentz case. In particular, when k is smaller than 8 (recall in Section 3 the Lorentz model can only correctly represent points when $k \leq 8$), then the difference in orders of magnitude for the Euclidean gradient descent is almost the same as the one for the Lorentz gradient descent. Hence, the Euclidean parametrization has similar performance to the Lorentz model in optimization when points are within a reasonable range. We empirically validate this point in tree embeddings experiments in Section 5.1.

Remark 4.3. (Guo et al., 2022) claimed that maps of the form $\mathbb{R}^n \xrightarrow{\exp_0} \mathbb{D}^n \xrightarrow{f} \mathbb{R}$ will incur gradient vanishing problem due to the fact that the Riemannian gradient of f will be small when the point is near the boundary of the Poincaré ball (see Lemma 3.4). However, this is only half of the story. In fact, the Jacobian of the map $\exp_0$ will reduce the power of δ from 2 to 1 and thus help alleviate the gradient vanishing problem (see Lemma 3.4, Theorem 4.2 and Appendix A.6 for more details).

4.2. Hyperplane Parametrization

The notion of hyperplanes in the hyperbolic space is a natural analogue of the notion of subspaces in the Euclidean spaces. This analogy results in that hyperbolic hyperplanes have been used for hyperbolic multinomial logistic regression (MLR) in (Ganea et al., 2018), in designing hyperbolic neural networks in (Shimizu et al., 2020), and of course, in designing hyperbolic SVM (Cho et al., 2019).

Consider $p \in \mathbb{L}^n, w \in T_p\mathbb{L}^n$, the Lorentz hyperplane passing through p and perpendicular to w is denoted by

$$H_{w,p} = \{x \in \mathbb{L}^n | \, [w, x] = 0\}. \quad (10)$$

Notice that p is implicit in the condition since w is on the tangent space of p. In hyperbolic MLR or hyperbolic neural networks, one needs to optimize over the choice of hyperplanes. Equation (10) gives us the opportunity to only use w for optimization. However, in order for w to be a feasible tangent vector, we need to impose the nonconvex restriction that $[w, w] > 0$ which is undesirable.

Instead, we derive a Euclidean parametrization of $H_{w,p}$ to get rid of the nonconvex constraint. This parametrization follows closely the Euclidean parametrization of hyperbolic features. We first parametrize p as in the previous section via $\exp_{\bar{0}}$ but with a flavour of polar coordinates. We then parametrize w as the parallel transport of a vector from $T_{\bar{0}}\mathbb{L}^n$ to $T_p\mathbb{L}^n$. Now, more precisely, let $a \in \mathbb{R}, z \in \mathbb{R}^n$, and set $\bar{z} := (0, z) \in T_{\bar{0}}\mathbb{L}^n$. We set

$$p := \exp_{\bar{0}}\left(a\frac{\bar{z}}{\|\bar{z}\|}\right) = \left(\cosh(a), \sinh(a)\frac{z}{\|z\|}\right). \quad (11)$$

Note that this equation is just the polar coordinate version of Equation (7). Then, we let

$$w := \mathrm{PT}_{\bar{0}\mapsto p}(\bar{z}) = (\sinh(a)\|z\|, \cosh(a)z) \quad (12)$$

Here $\mathrm{PT}_{x\mapsto y}$ denotes the parallel transport map along the unique geodesic from x to y and see a succinct derivation of the above formula in Appendix A.7.

Hence, the parametrized hyperplane becomes

$$\tilde{H}_{z,a} = \{x \in \mathbb{L}^n | \cosh(a)\langle z, x_r\rangle = \sinh(a)\|z\| x_0\},$$

where $x_r = (x_1, \dots, x_n)$ denotes the latter n components of $x = (x_0, x_1, \dots, x_n)$. The geometric meaning of this parametrization is as follows: the hyperplane is passing through a point p which is $|a|$ far away from the origin and along the direction z, and is perpendicular to a vector w which has length $\|z\|$.

Remark 4.4. (Shimizu et al., 2020) parametrized hyperplanes in the Poincaré ball for the purpose of reducing the number of training parameters from $2n$ to $n+1$ for hyperplanes in $\mathbb{D}^n$. Our derivation starts from the Lorentz model and is more succinct from the approach via the Poincaré ball. Eventually, we obtain essentially the formulas up to a small difference: The terms $\cosh(a)$ and $\sinh(a)$ in the formula above are replaced by $\cosh(2a)$ and $\sinh(2a)$ in (Shimizu et al., 2020). This difference is due to the mistake that they did not take into account the conformal factor $\lambda_0^2 = 4$ of $T_0\mathbb{D}^n$. See also Remark 4.1.

Remark 4.5 (What if $z = 0$?). The careful reader may notice that Equation (11) does not work for the case when $z = 0$. We note that however, the final formula Equation (12) works for the case when $z = 0$. When $z = 0$, the tangent vector w becomes the zero vector at p and thus the notion of hyperplane degraded to the whole space.

Now, we see there is no restriction on z and a and thus we get rid of the nonconvex condition $[w, w] > 0$ for parametrizing hyperplanes. This substantially simplifies the optimization process. Next we demonstrate how this parametrization can be used for hyperbolic SVM.

4.2.1. A New Formulation of Hyperbolic SVM

Support Vector Machine (SVM) is a classic machine learning model for both classification and regression by training a separating hyperplane. (Cho et al., 2019) first generalized SVM to datasets in the hyperbolic space for learning a separating hyperbolic hyperplane through the Lorentz model. Consider a training set $\{((x_i), (y_i))\}_{i=1}^n$ where $x_i \in \mathbb{L}^n, y_i \in \{1, -1\}, \forall i \in \{1, ..., n\}$. Then, the (soft-margin) Lorentz SVM (LSVM) can be formulated as

$$\min_{w \in \mathbb{R}^{n+1}, [w,w]>0} \frac{1}{2} \|w\|^2_{\mathbb{L}^n} + C \sum_{i=1}^{n} l_{\mathbb{L}^n}(-y_i[w, x_i])$$

where $l_{\mathbb{L}^n}(z) = \max(0, \operatorname{arcsinh}(1) - \operatorname{arcsinh}(z))$ is a variant of the hinge function that respects the Lorentz distance and C is a scalar controlling the tolerance of misclassification. This problem has both nonconvex constraint and nonconvex objective function and the authors propose using a projected gradient descent optimization.

As the vector w in the above formulation serves as the tangent vector determining a Lorentz hyperplane (cf. Equation (10)), we can then parametrize w via Equation (12). As w is the parallel transport of $\bar{z}$, we have that $\|w\|_{\mathbb{L}^n} = \|\bar{z}\|_{\mathbb{L}^n} = \|z\|$. In this way, we obtain the following minimization problem over parameters $z \in \mathbb{R}^n$ and $a \in \mathbb{R}$ without any nonconvex constraints.

$$\min_{z \in \mathbb{R}^n, a \in \mathbb{R}} \frac{1}{2} \|z\|^2 + C \sum_{i=1}^{n} l_{\mathbb{L}^n}(y_i(\sinh(a) \|z\| x_0 - \cosh(a)\langle z, x_r \rangle))$$

Empirical evaluation in Section 5.2 shows that this new SVM framework (LSVMPP) outperforms LSVM, possibly by escaping the local minima.

5. Experiments

5.1. Hyperbolic Embeddings

This subsection aims to empirically validate our theoretical results (Theorem 3.5 and Theorem 4.2) by demonstrating that both the Lorentz and the Euclidean (parametrized) models better leverage the hyperbolic structure than the Poincaré ball. To do so we embed trees in hyperbolic spaces. The embedding performance is measured by distortion, max distortion, and diameter. Distortion quantifies the fidelity of embedding w.r.t. the tree structure by simultaneously considering the expansion and contraction of the embedding. In a general setting, let $f : (M_R, d_R) \to (M_E, d_E)$ be an embedding map between two metric spaces, the distortion of the map is given by $\delta = \delta_{\text{contraction}} \cdot \delta_{\text{expansion}}$ where

$$\delta_{\text{contraction}} = \operatorname*{mean}_{x \neq y} \frac{d_R(x, y)}{d_E(x, y)} \; ; \; \delta_{\text{expansion}} = \operatorname*{mean}_{x \neq y} \frac{d_E(x, y)}{d_R(x, y)}$$

Max distortion $\delta^{\max}$ is defined similarly by replacing the mean with the supremum.

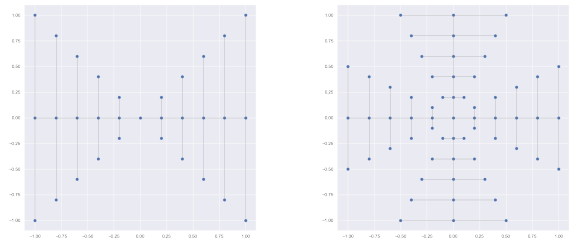

Figure 1. Simulated Tree 1 and Tree 2 in $\mathbb{R}^2$. The 2D coordinates of each node are features and pairwise distances are computed through shortest path distance on the connected graph.

Datasets We simulated tree datasets S in $\mathbb{R}^2$ with varying degrees of complexity whose distances are equally weighted graph distances. Figure 1 lists two example trees with more in Figure 5. A detailed discussion on the impact of performance by its shape can be found in the Appendix B.1.

Experiments We optimize the following loss function to minimize embedding distortion:

$$\mathcal{L}(\theta) = \frac{1}{|S|(|S| - 1)} \sum_{x \neq y \in S} \left(\frac{d_E(x, y)}{z_E} - \frac{d_R(x, y)}{z_R} \right)^2$$

where θ collects the parameters in the network, d_E denotes the embedding distance and d_R denotes the original tree distance, and z_E, z_R are averages of d_E and d_R, respectively. For $\mathbb{L}^2$ and $\mathbb{D}^2$, the features to update are the results of the respective exponential maps F_L, F_D of each dataset (treating Euclidean data points as in the tangent space of the origin). For the Euclidean model $\mathbb{E}^2$, the features to update are the raw $\mathbb{R}^2$ dataset, but the pairwise distances are computed after F_L or equivalently F_D. We use Riemannian SGD (Becigneul & Ganea, 2018) for hyperbolic models and SGD for the Euclidean model, fixing a learning rate of 1 and train for 30000 epochs.

Results We report the metrics after the final epoch in Table 3. Here raw manifold refers to using Euclidean distances as embedding distances against tree distances to measure distortions (i.e. original distortions). In both cases, the benefit of Euclidean parametrization to train hyperbolic models is clear: the Euclidean model achieves the lowest average distortions and max distortion, although the performances of Lorentz and Euclidean are comparable as expected. Furthermore, the diameters of the resulting embeddings for the Lorentz and Euclidean models are much larger than the Poincaré embeddings. This validates that Poincaré tends to get stuck using Riemannian optimization methods.

Table 1. Average distortion δ, max distortion $\delta^{\max}$, and diameter d by Dataset created in Figure 1 and Manifold

tree	manifold	δ	$\delta^{\max}$	d
1	raw	1.1954	10.6062	2.8284
	$\mathbb{D}^2$	1.0462	4.0546	4.8740
	$\mathbb{L}^2$	1.0176	2.4511	10.1827
	$\mathbb{E}^2$	**1.0157**	**2.3158**	**10.9019**
2	raw	1.1186	14.1421	2.2361
	$\mathbb{D}^2$	1.0589	7.1435	4.9757
	$\mathbb{L}^2$	1.0180	3.2803	10.6719
	$\mathbb{E}^2$	**1.0134**	**2.7408**	**11.4504**

Figure 2 visualizes the embeddings of Tree 1 and Tree 2 in Figure 1 (see also Figure 6 and Figure 7 in Appendix B for visualizations of embeddings for all simulated trees). Lorentz features are stereographically projected to the Poincaré ball for comparisons. Dimmer points are closer to the edge of the input space while darker points are more at the center. The red point is the root or center of the tree. Poincaré results are seemingly not as spread out as the other two methods, and resembling the early stage in the optimization process of Euclidean and Lorentz models.

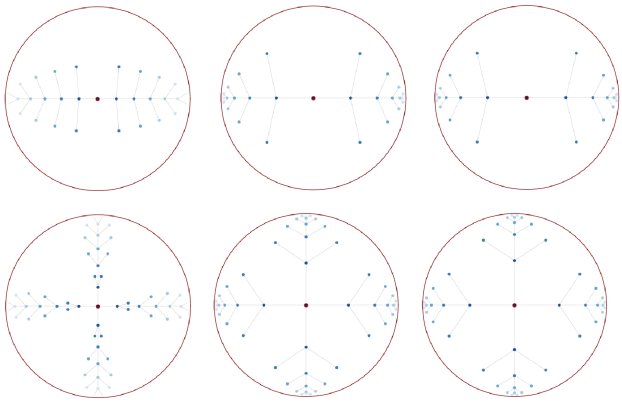

Figure 2. Embedding of Tree 1 (top row) and Tree 2 (bottom row) at the final epoch with Poincaré (left), Lorentz (mid), and Euclidean parametrization (right).

Finally, in Figure 3, we directly compare the median norms of Riemannian gradient from respective manifolds along different training epochs, whose median is taken across different embedding points. We visualize two ratios, one between gradient norms of $\mathbb{L}^2$ and $\mathbb{E}^2$ (blue) and the other between $\mathbb{D}^2$ and $\mathbb{E}^2$ (red). In both cases, Poincaré graidents have much smaller a norm than its competitors. This further confirms its optimization disadvantage.

Figure 3. Median Riemannian Gradient Norm Ratios by epoch for Tree 1 (left) and Tree 2 (right): Poincaré gradients have significantly smaller norms than others

5.2. Hyperbolic SVM Models

We tested the performances of all models in multi-class classification tasks using the following six simulated datasets and six real datasets.

Synthetic Datasets We simulated two sets of data: *Gaussian mixtures* and *explicit trees*. For Gaussian mixtures, we simulate each dataset using 1200 points on the Poincaré disk $\mathbb{D}^2$ following the settings in (Cho et al., 2019). Specifically, we first generate a Gaussian mixture on $\mathbb{R}^2$, whose k centroids follow $N(\mathbf{0}, 1.5\mathbb{I}_2)$ equipped with $N(\mathbf{0}, \mathbb{I}_2)$ noise. Each of the k clusters is given a unique label and has the same number of points. We then use the map F_D to map Euclidean features to Poincare features. We tested performances on $k \in \{3, 5, 10\}$. For each data set of explicit trees, we initialize a tree in $\mathbb{R}^2$ and then embed the tree into the hyperbolic space using Euclidean parametrization as described in Section 5.1. Then we randomly select a node and treat all of its descendants as positive class and negative class otherwise (i.e. $k = 2$). This subtree-classification also resembles the WordNet subtree classification presented in (Cho et al., 2019). See Appendix C.4 for more details.

Real Datasets We tested the performances on three datasets: CIFAR-10 (Krizhevsky et al., 2009), fashion-MNIST (Xiao et al., 2017), Paul Myeloid Progenitors developmental dataset (Paul et al., 2015), Olsson Single-Cell RNA sequencing dataset (Olsson et al., 2016), Krumsiek Simulated Myeloid Progenitors (Krumsiek et al., 2011), and Moignard blood cell developmental trace from single-cell gene expression (Moignard et al., 2015). CIFAR-10 and Fashion-MNIST each contain images of 10 different types of objects. Other datasets are rooted in biological applications: Paul has 19 cell types developed from the myeloid progenitors;

Krumsiek likewise but with a different latent structure and 11 classes; Olsson is a smaller dataset with RNA-seq data of 8 types; Moignard has 7 classes for different blood cell developmental stages. Each dataset is embedded into $\mathbb{D}^2$ with the default curvature 1 using the technique introduced in (Khrulkov et al., 2020; Nickel & Kiela, 2017). We visualize three of the tested datasets in Figure 4.

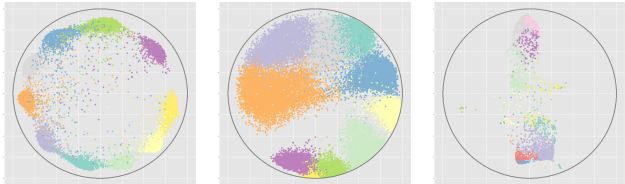

Figure 4. CIFAR, Fashion-MNIST, and Paul datasets Poincaré visualization. Each color correspond to one label class.

Results For each dataset, we fix a train-test split and run 5 times. In each run, we use a one-vs-all approach to train and use Platt scaling (Platt et al., 1999) to give final predictions based on output probabilities. The average multiclass classification accuracy and macro F1 scores are reported. We compare our model (LSVMPP) against the Lorentz SVM (LSVM), the Euclidean SVM (ESVM) and a SVM with precomputed reference points based on the Poincaré ball (PSVM) (Chien et al., 2021). See Appendix C.3 for and a detailed discussion on implementation and respective optimal hyperparameters, Table 2 for average accuracy and F1 for selected datasets in Figure 4, and Appendix C.5 for experiment results on more datasets.

Although small fluctuations exist, our model outperforms other available methods in most of the simulated and real datasets. Note that in all cases, Euclidean SVM already has decent performances, unlike the unsatisfactory performances reported in (Chien et al., 2021) where ESVM models are mistakenly set to not have intercepts. The good performance by ESVM may be attributed to the powerful scaling method that universally applies to datasets well characterized by the log odds of positive samples, and this assumption may still hold on datasets with a latent hierarchical structure. It is also worth pointing out that the previous formulation of Lorentz SVM (Cho et al., 2019) also uses Platt scaling on Lorentz SVM output but did not scale it using arcsinh to morph it into a proxy of hyperbolic distance, which is more aligned with the Euclidean SVM output. Empirically we witness a marginal increase in average accuracy using Platt scaling with the output of arcsinh instead of the raw alternatives (see Appendix C.2.1 for more details).

Lastly, notice that the previously leading model, PSVM, does not outperform ESVM in simple simulated cases. This matches our expectations in that the reference points in PSVM are separately learned using an unsupervised method (Hyperbolic Graham Search) to locate the midpoint between two convex hulls, independent of the hyperplane training process. This step is sensitive to the train test split and does not guarantee that PSVM could identify the max-margin separating hyperplane even in the training set. All other methods, if they converge, could indeed find the max-margin separating hyperplane in respective manifolds.

Table 2. Mean accuracy and macro F1 score of datasets in Figure 4.

	algorithm	accuracy (%)	F1
CIFAR-10	ESVM	91.88 ± 0.00	0.9191 ± 0.00
	LSVM	91.88 ± 0.00	0.9189 ± 0.00
	PSVM	91.81 ± 0.00	0.9182 ± 0.00
	LSVMPP	$\mathbf{91.96 \pm 0.00}$	$\mathbf{0.9197 \pm 0.00}$
fashion-MNIST	ESVM	86.37 ± 0.00	0.8665 ± 0.00
	LSVM	71.59 ± 0.07	0.6588 ± 0.08
	PSVM	86.57 ± 0.00	0.8665 ± 0.00
	LSVMPP	$\mathbf{89.49 \pm 0.00}$	$\mathbf{0.8955 \pm 0.00}$
paul	ESVM	55.05 ± 0.00	0.4073 ± 0.00
	LSVM	58.36 ± 0.07	0.4517 ± 0.00
	PSVM	55.25 ± 0.00	0.3802 ± 0.00
	LSVMPP	$\mathbf{62.64 \pm 0.05}$	$\mathbf{0.5024 \pm 0.00}$

6. Discussions

In this paper, we have analyzed the representation limitations and numerical stability of optimization for two popular models and a Euclidean parametrization of the hyperbolic space under a 64-bit arithmetic system. We note that however, while the Euclidean parameterization has no limitation for point representation, computation of certain functions induced from hyperbolic spaces, such as the hyperbolic distance between points, may still lead to numerical problems and we leave this for future study. Another potential future direction is to investigate the use of different accelerated gradient descent methods on these models. A recent study (Hamilton & Moitra, 2021) has shown that the negative curvature inherent to the hyperbolic space can impede the acceleration of the gradient descent process in a Nestorov-like way, citing the exponential volume growth property as a cause for the loss of information from past gradients. It would be interesting to explore in the future whether the Euclidean parametrization proposed in our paper could overcome this obstruction and improve the convergence rate of optimization problems in hyperbolic spaces.

Code for reproducing our experiments is available at https://github.com/yangshengaa/stable-hyperbolic.

Acknowledgements

This work is partially supported by NSF-CCF-2112665, NSF-CCF-2217033.

References

Becigneul, G. and Ganea, O.-E. Riemannian adaptive optimization methods. In *International Conference on Learning Representations*, 2018.

Bonnabel, S. Stochastic gradient descent on Riemannian manifolds. *IEEE Transactions on Automatic Control*, 58 (9):2217–2229, 2013.

Chamberlain, B. P., Hardwick, S. R., Wardrope, D. R., Dzogang, F., Daolio, F., and Vargas, S. Scalable hyperbolic recommender systems. *arXiv preprint arXiv:1902.08648*, 2019.

Chen, W., Han, X., Lin, Y., Zhao, H., Liu, Z., Li, P., Sun, M., and Zhou, J. Fully hyperbolic neural networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5672–5686, 2022.

Chien, E., Pan, C., Tabaghi, P., and Milenkovic, O. Highly scalable and provably accurate classification in Poincaré balls. In *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 61–70. IEEE, 2021.

Cho, H., DeMeo, B., Peng, J., and Berger, B. Large-margin classification in hyperbolic space. In *The 22nd international conference on artificial intelligence and statistics*, pp. 1832–1840. PMLR, 2019.

Ganea, O., Bécigneul, G., and Hofmann, T. Hyperbolic neural networks. *Advances in neural information processing systems*, 31, 2018.

Gao, S., Mishne, G., and Scheinost, D. Poincaré embedding reveals edge-based functional networks of the brain. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2020*, pp. 448–457. Springer, 2020.

Graham, R. L. An efficient algorithm for determining the convex hull of a finite planar set. *Info. Pro. Lett.*, 1: 132–133, 1972.

Guo, Y., Wang, X., Chen, Y., and Yu, S. X. Clipped hyperbolic classifiers are super-hyperbolic classifiers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11–20, 2022.

Hamilton, L. and Moitra, A. A no-go theorem for robust acceleration in the hyperbolic plane. *Advances in Neural Information Processing Systems*, 34:3914–3924, 2021.

Khrulkov, V., Mirvakhabova, L., Ustinova, E., Oseledets, I., and Lempitsky, V. Hyperbolic image embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6418–6428, 2020.

Kochurov, M., Karimov, R., and Kozlukov, S. Geoopt: Riemannian optimization in pytorch. *arXiv preprint arXiv:2005.02819*, 2020.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Krumsiek, J., Marr, C., Schroeder, T., and Theis, F. J. Hierarchical differentiation of myeloid progenitors is encoded in the transcription factor network. *PloS one*, 6(8):e22649, 2011.

Law, M., Liao, R., Snell, J., and Zemel, R. Lorentzian distance learning for hyperbolic representations. In *International Conference on Machine Learning*, pp. 3672–3681. PMLR, 2019.

López, F. and Strube, M. A fully hyperbolic neural model for hierarchical multi-class classification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 460–475, 2020.

López, F., Heinzerling, B., and Strube, M. Fine-grained entity typing in hyperbolic space. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pp. 169–180, 2019.

Mathieu, E., Le Lan, C., Maddison, C. J., Tomioka, R., and Teh, Y. W. Continuous hierarchical representations with Poincaré variational auto-encoders. *Advances in neural information processing systems*, 32, 2019.

Moignard, V., Woodhouse, S., Haghverdi, L., Lilly, A. J., Tanaka, Y., Wilkinson, A. C., Buettner, F., Macaulay, I. C., Jawaid, W., Diamanti, E., et al. Decoding the regulatory network of early blood development from single-cell gene expression measurements. *Nature biotechnology*, 33(3): 269–276, 2015.

Nickel, M. and Kiela, D. Poincaré embeddings for learning hierarchical representations. *Advances in neural information processing systems*, 30, 2017.

Nickel, M. and Kiela, D. Learning continuous hierarchies in the Lorentz model of hyperbolic geometry. In *International Conference on Machine Learning*, pp. 3779–3788. PMLR, 2018.

Olsson, A., Venkatasubramanian, M., Chaudhri, V. K., Aronow, B. J., Salomonis, N., Singh, H., and Grimes, H. L. Single-cell analysis of mixed-lineage states leading to a binary cell fate choice. *Nature*, 537(7622):698–702, 2016.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Paul, F., Arkin, Y., Giladi, A., Jaitin, D. A., Kenigsberg, E., Keren-Shaul, H., Winter, D., Lara-Astiaso, D., Gury, M., Weiner, A., et al. Transcriptional heterogeneity and lineage commitment in myeloid progenitors. *Cell*, 163 (7):1663–1677, 2015.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Peng, W., Varanka, T., Mostafa, A., Shi, H., and Zhao, G. Hyperbolic deep neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Platt, J. et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

Sala, F., De Sa, C., Gu, A., and Ré, C. Representation tradeoffs for hyperbolic embeddings. In *International conference on machine learning*, pp. 4460–4469. PMLR, 2018.

Sarkar, R. Low distortion Delaunay embedding of trees in hyperbolic plane. In *International Symposium on Graph Drawing*, pp. 355–366. Springer, 2011.

Shimizu, R., Mukuta, Y., and Harada, T. Hyperbolic neural networks++. In *International Conference on Learning Representations*, 2020.

Skopek, O., Ganea, O.-E., and Bécigneul, G. Mixed-curvature variational autoencoders. In *International Conference on Learning Representations*, 2019.

Ungar, A. A. Hyperbolic trigonometry and its application in the Poincaré ball model of hyperbolic geometry. *Computers & Mathematics with Applications*, 41(1-2):135–147, 2001.

Wilson, B. and Leimeister, M. Gradient descent in hyperbolic space. *arXiv preprint arXiv:1805.08207*, 2018.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yu, T. and De Sa, C. M. Numerically accurate hyperbolic embeddings using tiling-based models. *Advances in Neural Information Processing Systems*, 32, 2019.

Zhu, Y., Zhou, D., Xiao, J., Jiang, X., Chen, X., and Liu, Q. Hypertext: Endowing fasttext with hyperbolic geometry. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1166–1171, 2020.

A. Proofs

A.1. Proof of Proposition 3.1

Note that $\operatorname{arccosh}(x) = \ln(x + \sqrt{x^2 - 1})$. This implies that

$$\operatorname{arccosh}\left(1 + \frac{2\|x\|^2}{1 - \|x\|^2}\right) = \operatorname{arccosh}\left(\frac{1 + \|x\|^2}{1 - \|x\|^2}\right) = \ln\left(\frac{1 + \|x\|^2}{1 - \|x\|^2} + \sqrt{\frac{(1 + \|x\|^2)^2}{(1 - \|x\|^2)^2} - 1}\right) = \ln\left(\frac{1 + \|x\|^2 + 2\|x\|}{1 - \|x\|^2}\right),$$

and then $\ln\left(\frac{1+\|x\|^2+2\|x\|}{1-\|x\|^2}\right) = \ln\left(\frac{1+\|x\|}{1-\|x\|}\right) = \ln(1 + \|x\|) - \ln(1 - \|x\|)$.

By using the formula for the Poincaré distance between 0 and x, we have that

$$\begin{aligned} d_{\mathbb{D}^n}(0, x) &= \operatorname{arccosh}\left(1 + \frac{2\,\|x\|^2}{1 - \|x\|^2}\right) \\ &= \ln(1 + \|x\|) - \ln(1 - \|x\|) \\ &= \ln(2 - 10^{-k}) - \ln(10^{-k}) \\ &= \ln(10)k + \ln(2) + \ln(1 - (2 * 10^k)^{-1}) \\ &= \ln(10)k + \ln(2) + O(10^{-k}). \end{aligned}$$

A.2. Proof of Proposition 3.2

By using the formula for the Lorentz distance between $\bar{0}$ and x, we have that

$$\begin{aligned} d_{\mathbb{L}^n}(\bar{0}, x) &= \operatorname{arccosh}(-[\bar{0}, x]) \\ &= \operatorname{arccosh}(x_0) = \ln\left(x_0 + \sqrt{x_0^2 - 1}\right) \\ &= \ln(x_0) + \ln\left(1 + \sqrt{1 - 1/x_0^2}\right) \\ &= \ln(x_0) + \ln\left(2 + \frac{1}{2x_0^2} + O(x_0^{-4})\right) \\ &= \ln(10)k + \ln(2) + O(10^{-2k}). \end{aligned}$$

A.3. Proof of Proposition 3.3

Given a Riemannian isometry $\iota : M \to N$ between two manifolds, for any $p \in M$, one has that

$$\iota \circ \exp_p = \exp_{\iota(p)} \circ D\iota_p,$$

where $D\iota_p : T_pM \to T_pN$ denotes the differential at p.

Now, by the Riemannian chain rule for gradient, one has that

$$\nabla_{\mathbb{D}^n} f(x) = \nabla_{\mathbb{D}^n}(g \circ \varphi)(x) = (D\varphi_x)^* \circ \nabla_{\mathbb{L}^n} g(y),$$

where $y = \varphi(x)$. Since φ is an isometry, one has that $(D\varphi_x)^* = D\psi_y : T_y\mathbb{L}^n \to T_x\mathbb{D}^n$, where $\psi := \varphi^{-1}$. In this way, $D\psi_y(-\eta\nabla_{\mathbb{L}^n} g(y)) = -\eta\nabla_{\mathbb{D}^n} f(x)$ and thus

$$\psi(\exp_y(-\eta\nabla_{\mathbb{L}^n} g(y)) = \exp_{\psi(y)}(D\psi_y(-\eta\nabla_{\mathbb{L}^n} g(y))) = \exp_x(-\eta\nabla_{\mathbb{D}^n} f(x)).$$

A.4. Proof of Lemma 3.4

The Poincaré gradient $\nabla_{\mathbb{D}^n} f$ at y can be computed explicitly as follows:

$$\nabla_{\mathbb{D}^n} f(x) = \lambda_x^{-2}\nabla f(x) = \frac{(1 - \|x\|^2)^2}{4}(\partial_1 f(x), \ldots, \partial_n f(y))^{\mathrm{T}}.$$

Hence,

$$\|\nabla_{\mathbb{D}^n} f(x)\| = \Omega\left(\frac{(1-\|x\|^2)^2}{4}\|\nabla f(x)\|\right).$$

As for the Lorentz gradient, we apply the following lemma

Lemma A.1. *For any $x \in \mathbb{D}^n$ and any $v = (v_1, \ldots, v_n) \in T_x\mathbb{D}^n$, by letting $y := \varphi(x) \in \mathbb{L}^n$, we can write down the image of v under the differential $D\varphi$ explicitly as follows:*

$$D\varphi(v) = \left(\sum_{i=1}^n y_i v_i(1+y_0), v_1(1+y_0) + \sum_i y_i v_i \cdot y_1, \ldots, v_n(1+y_0) + \sum_i y_i v_i \cdot y_n\right). \tag{13}$$

Proof of Lemma A.1. The differential of $\psi : \mathbb{L}^n \to \mathbb{D}^n$, the inverse of φ, was already derived in Equation (9) of (Wilson & Leimeister, 2018): for $w = (w_0, \ldots, w_n) \in T_y\mathbb{L}^n$, and for any $1 \le i \le n$

$$(D\psi(w))_i = \frac{1}{1+y_0}\left(w_i - \frac{y_i w_0}{y_0+1}\right).$$

We then simply verify that the map defined in Equation (13) is the inverse of $D\psi$. Let w be the right hand side of Equation (13). Then, for any $1 \le i \le n$

$$\begin{aligned}(D\psi(w))_i &= \frac{1}{1+y_0}\left(w_i - \frac{y_i w_0}{y_0+1}\right)\\ &= \frac{1}{1+y_0}\left(v_i(1+y_0) + \sum_j y_j v_j \cdot y_i - \frac{y_i}{y_0+1}\sum_{j=1}^n y_j v_j(1+y_0)\right)\\ &= v_i.\end{aligned}$$

Hence, Equation (13) holds. □

As the map $\varphi : \mathbb{D}^n \to \mathbb{L}^n$ is an isometry, by the lemma above, the Lorentz gradient at y can be written down explicitly:

$$\begin{aligned}\nabla_{\mathbb{L}^n} g(y) &= D\varphi(\nabla_{\mathbb{D}^n} f(x)) = D\varphi\left(\frac{(1-\|x\|^2)^2}{4}\nabla f(x)\right) = D\varphi\left(\frac{1}{(1+y_0)^2}\nabla f(x)\right)\\ &= \left(\frac{\sum_{i=1}^n y_i \partial_i f(x)}{1+y_0}, \frac{\partial_1 f(x)(1+y_0) + \sum_i y_i \partial_i f(x)\cdot y_1}{(1+y_0)^2}, \ldots, \frac{\partial_n f(x)(1+y_0) + \sum_i y_i \partial_i f(x)\cdot y_n}{(1+y_0)^2}\right)^{\mathrm{T}}.\end{aligned}$$

Note that we use column vector representation in the above equation to indicate that the vector denotes a gradient. In the third equation, we used the fact that $1 + y_0 = 1 + \frac{1+\|x\|^2}{1-\|x\|^2} = \frac{2}{1-\|x\|^2}$. Then,

$$\|\nabla_{\mathbb{L}^n} g(y)\| = \frac{\sqrt{2(\sum y_i \partial_i f(x))^2 + \sum(\partial_i f(x))^2)}}{1+y_0} \le \frac{\sqrt{2y_0^2 - 1}}{1+y_0}\|\nabla f(x)\| = O(\|\nabla f(x)\|).$$

A.5. Proof of Theorem 3.5

Recall that in the Poincaré ball, the exponential map can be expressed as follows:

$$\exp_x(v) = x \oplus \tanh\left(\frac{\lambda_x}{2}\|v\|\right)\frac{v}{\|v\|},$$

where $\lambda_x = \frac{2}{1-\|x\|^2}$ and

$$x \oplus y = \frac{(1 + 2\langle x, y\rangle + \|y\|^2)x + (1-\|x\|^2)y}{1 + 2\langle x, y\rangle + \|x\|^2\|y\|^2}. \tag{14}$$

When $x = (1-\delta, 0, \dots, 0)$, $\lambda_x = \frac{2}{\delta(2-\delta)}$. We let $E := \|\nabla f(x)\|$. Then,

$$\begin{aligned}
\exp_x(-\eta \nabla_{\mathbb{D}^n} f(x)) &= x \oplus \tanh\left(\frac{\lambda_x \eta}{2} \|\nabla_{\mathbb{D}^n} f(x)\|\right) \frac{-\nabla_{\mathbb{D}^n} f(x)}{\|\nabla_{\mathbb{D}^n} f(x)\|} \\
&= x \oplus \tanh\left(\frac{\lambda_x \eta}{2} \|\nabla_{\mathbb{D}^n} f(x)\|\right) \frac{-\nabla_{\mathbb{D}^n} f(x)}{\|\nabla_{\mathbb{D}^n} f(x)\|} \\
&= x \oplus \tanh\left(\frac{\eta}{2\lambda_x} \|\nabla f(x)\|\right) \frac{-\nabla f(x)}{\|\nabla f(x)\|} \\
&= x \oplus \tanh\left(\frac{\eta E \delta(2-\delta)}{4}\right) \frac{-\nabla f(x)}{\|\nabla f(x)\|} \\
&= x \oplus \left(\frac{2-\delta}{4} \eta E \delta + O((\eta E \delta)^3)\right) \frac{-\nabla f(x)}{\|\nabla f(x)\|}.
\end{aligned}$$

We use the Taylor expansion of $\tanh$ at 0 in the last equation. Now, using Equation (14), we have that

$$\begin{aligned}
& x \oplus \left(\frac{2-\delta}{4} \eta E \delta + O((\eta E \delta)^3)\right) \frac{-\nabla f(x)}{\|\nabla f(x)\|} \\
=& \frac{(1 - 2(1-\delta)(\frac{2-\delta}{4}\eta E\delta + O((\eta E\delta)^3)) + O((\eta E\delta)^2))(1-\delta, 0, \dots, 0) + \delta(2-\delta)(\frac{2-\delta}{4}\eta E\delta + O((\eta E\delta)^2))(1, 0, \dots, 0)}{1 - 2(1-\delta)(\frac{2-\delta}{4}\eta E\delta + O((\eta E\delta)^3)) + (1-\delta)^2 O((\eta E\delta)^2)} \\
=& \frac{\big(1 - 2(1-\delta)\frac{2-\delta}{4}\eta E\delta + O((\eta E\delta)^2))(1-\delta) + O(\eta E\delta^2), 0, \dots, 0\big)}{1 - 2(1-\delta)\frac{2-\delta}{4}\eta E\delta + O((\eta E\delta)^2)} \\
=& (1 - \delta + O(\eta E \delta^2), 0, \dots, 0).
\end{aligned}$$

As for the Lorentz model, we have that

$$\begin{aligned}
& \exp_y(-\eta \nabla_{\mathbb{L}^n} g(y)) = \cosh(\eta \|\nabla_{\mathbb{L}^n} g(y)\|_{\mathbb{L}^n}) y + \sinh(\eta \|\nabla_{\mathbb{L}^n} g(y)\|_{\mathbb{L}^n}) \frac{-\nabla_{\mathbb{L}^n} g(y)}{\|\nabla_{\mathbb{L}^n} g(y)\|_{\mathbb{L}^n}} \\
=& \cosh(\eta \|\nabla_{\mathbb{D}^n} f(x)\|_{\mathbb{D}^n}) y + \sinh(\eta \|\nabla_{\mathbb{D}^n} f(x)\|_{\mathbb{D}^n}) \frac{-\nabla_{\mathbb{L}^n} g(y)}{\|\nabla_{\mathbb{D}^n} f(x)\|_{\mathbb{D}^n}} \\
=& \cosh(\frac{(2-\delta)}{2}\eta E\delta)\big(\frac{2-2\delta+\delta^2}{\delta(2-\delta)}, \frac{2-2\delta}{\delta(2-\delta)}, 0, \dots, 0\big) \\
& - \frac{\sinh(\frac{(2-\delta)}{2}\eta E\delta)}{\frac{(2-\delta)}{2}E\delta}\big((1-\delta)\partial_1 f(x), \frac{2-2\delta+\delta^2}{2}\partial_1 f(x), 0, \dots, 0\big) \\
=& (1 + O((\eta E\delta)^2))\big(\frac{2-2\delta+\delta^2}{\delta(2-\delta)}, \frac{2-2\delta}{\delta(2-\delta)}, 0, \dots, 0\big) + (\eta + O(\eta^2 (E\delta)^3))\big((1-\delta)E, \frac{2-2\delta+\delta^2}{2}E, 0, \dots, 0\big) \\
=& \big(\frac{1}{\delta} - \frac{1}{2} + \eta E + O(\eta E\delta), \frac{1}{\delta} - \frac{1}{2} + \eta E + O(\eta E\delta), 0, \dots, 0\big)
\end{aligned}$$

Finally, we substitute δ with 10^{-k} to obtain

$$\begin{aligned}
\exp_x(-\eta \nabla_{\mathbb{D}^n} f(x)) &= (1 - 10^{-k} + O(\eta E 10^{-2k}), 0, \dots, 0), \\
\exp_y(-\eta \nabla_{\mathbb{L}^n} g(y)) &= (10^k - \frac{1}{2} + \eta E + O(\eta E 10^{-k}), 10^k - \frac{1}{2} + \eta E + O(\eta E 10^{-k}), 0, \dots, 0).
\end{aligned}$$

A.6. Proof of Theorem 4.2

Similarly as in the proof of Appendix A.5, we denote $E := \|\nabla f(x)\|$. We first calculate the gradient ∇h. We let J_{F_D} denote the Jacobian of F_D. Then, the differential of h can be computed via the chain rule

$$Dh(z) = Df(x) \cdot J_{F_D}(z)$$

Therefore, as for the gradient, we have that

$$\nabla h(z) = (Dh(z))^{\mathrm{T}} = (\mathrm{J}_{F_D}(z))^{\mathrm{T}} \nabla f(x), \tag{15}$$

where we use the fact that $\nabla f(x) = (Df(x))^{\mathrm{T}}$.

Remark A.2. Of course, one also can apply the chain rule formula for Riemannian gradient directly as follows

$$\nabla h(z) = (DF_D(z))^* \cdot \nabla_{\mathbb{D}^n} f(x),$$

where $(DF_D(z))^*$ denotes the adjoint of the differential map $DF_D(z) : T_z\mathbb{R}^n \to T_x\mathbb{D}^n$, which is represented by a matrix. Given that the metric on $T_x\mathbb{D}^n$ is $\lambda_x^2 h_x$ where h_x denotes the Euclidean metric, we have that the matrix representation of $(DF_D(z))^*$ is $(\mathrm{J}_{F_D}(z))^{\mathrm{T}}\lambda_x^2$. Hence

$$\nabla h(z) = (\mathrm{J}_{F_D}(z))^{\mathrm{T}}\lambda_x^2 \cdot \underbrace{\lambda_x^{-2}\nabla f(x)}_{A} = (\mathrm{J}_{F_D}(z))^{\mathrm{T}}\nabla f(x).$$

This agrees with Equation (15). (Guo et al., 2022) claimed that functions of the form of h, i.e., the form $\mathbb{R}^n \xrightarrow{\exp_0} \mathbb{D}^n \to \mathbb{R}$, will suffer severe vanishing gradient issue since the term A has a coefficient $\lambda_x^{-2} = \frac{(1-\|x\|^2)^2}{4}$ vanishing as x is close to the boundary of $\mathbb{D}^n$. However, as one can see clearly from the above formula, the coefficient $\frac{(1-\|x\|^2)^2}{4}$ will be canceled when considering the map F_D (or equivalently speaking, the exponential map $\exp_0$). Hence, in order to correctly analyze the magnitude of ∇h, one needs to compute the Jacobian matrix which is what we do next in a special case.

Let $w := \operatorname{arctanh}(1-\delta)$. Then, $z = (2w, 0, \ldots, 0)$ and the Jacobian of $F_D : \mathbb{R}^n \to \mathbb{D}^n$ at $z = (2w, 0, \ldots, 0)$ is

$$\mathrm{J}_{F_D}(z) = \frac{\tanh(w)}{2w} * \mathrm{I}_{n\times n} + \begin{bmatrix} \frac{1}{2}(1-\tanh^2(w) - \frac{\tanh(w)}{w}) & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 \end{bmatrix}.$$

Hence,

$$\begin{aligned}\nabla h(z) = (\mathrm{J}_{F_D}(z))^{\mathrm{T}}\nabla f(x) &= \left(\frac{\tanh(w)}{2w} * \mathrm{I}_{n\times n} + \begin{bmatrix} \frac{1}{2}(1-\tanh^2(w) - \frac{\tanh(w)}{w}) & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 \end{bmatrix}\right) \begin{bmatrix} \partial_1 f(x) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ &= \left[\frac{\partial_1 f(x)}{2}(1-\tanh^2(w)), 0, \ldots, 0\right]^{\mathrm{T}}.\end{aligned}$$

Therefore, the first component of $z - \eta\nabla h(z)$ is such that

$$\begin{aligned} & 2w + \frac{\eta E}{2}(1-\tanh^2(w)) \\ =& 2w + \frac{\eta E}{2}(1-(1-\delta)^2) \\ =& \ln\left(\frac{2}{\delta}\right) - \frac{\delta}{2} + O(\delta^2) + \frac{\eta E}{2}(2\delta - \delta^2) \\ =& \ln\left(\frac{2}{\delta}\right) + (\eta E - 1/2)\delta + O(\delta^2). \end{aligned}$$

Note that in the second equation we use the expansion that

$$\operatorname{arctanh}(1-\delta) = \frac{1}{2}\ln\left(\frac{2}{\delta}\right) - \frac{\delta}{4} + O(\delta^2).$$

Finally, we substitute δ with 10^{-k} to obtain

$$z - \eta\nabla h(z) = \ln(10)k + \ln(2) + (\eta\,\|\nabla f(x)\| - 1/2)10^{-k} + O(10^{-2k}).$$

A.7. Derivation of Equation (12)

In the hyperbolic space $\mathbb{H}^n$, any two points can be connected by a unique geodesic. Using the Lorentz model, for any $x, y \in \mathbb{L}^n$, the parallel transport map along the geodesic connecting x and y can be written down explicitly as follows: for $v \in T_x\mathbb{L}^n$, one has that

$$\mathrm{PT}_{x\mapsto y}(v) = v + \frac{[y,v]}{1-[x,y]}(x+y).$$

As usual we define $\bar{0} := [1, 0, ..., 0] \in \mathbb{R}^{n+1}$ to be the origin of the Lorentz model. Then

$$\begin{aligned}
w = \mathrm{PT}_{\bar{0}\mapsto p}(\bar{z}) &= \bar{z} + \frac{[p,\bar{z}]}{1-[\bar{0},p]}(\bar{0}+p) \\
&= (0,z) + \frac{\sinh(a)\,\|z\|}{1+\cosh(a)}\left(1+\cosh(a), \sinh(a)\frac{z}{\|z\|}\right) \\
&= \left(\sinh(a)\,\|z\|\,, z + \frac{\sinh^2(a)}{1+\cosh(a)}z\right) \\
&= \left(\sinh(a)\,\|z\|\,, z + \frac{\cosh^2(a)-1}{1+\cosh(a)}z\right) \\
&= (\sinh(a)\,\|z\|\,, \cosh(a)z)\,.
\end{aligned}$$

B. Tree Optimization

In this section, we provide more details on the tree optimization experiments.

B.1. Synthetic Data Generating Processing

We prefer to initialize, center, and normalize trees with Euclidean coordinates in a prescribed bounded region, serving as a data preprocessing procedure. Here we fit every simulated tree to be within a unit square. This ensures numerical stability in the initial exponential map into either $\mathbb{D}^n$ or $\mathbb{L}^n$, and empirically has lower embedding distortion than the same ones with larger scales. The reason is that if the data is not centered or of a large scale, the subsequent exponential map brings features to much more distorted positions and poses challenges in the training steps.

B.2. Full Results

The Euclidean initialization of all tested trees are visualized in Figure 5. In Table 3, we present all embedding results in distortion δ, max distortion $\delta^{\max}$, and embedding diameters d.

The hierarchy that the Euclidean parametrization is comparable with and slightly better than the Lorentz model and that the Lorentz model is better than the Poincaré ball persists in all tested datasets. We also visualize all embeddings at their final training stage in Figure 6 and Figure 7. Note that Poincaré embeddings in general are more contracted than their alternatives, further demonstrating its optimization disadvantage.

C. More on SVM

In this section, we provide more details of our experiments on SVM.

C.1. Overview of SVM in Binary Classification

Support Vector Machine (SVM) is a machine learning task for both classification and regression by training a separating hyperplane. We first provide a brief overview of the Euclidean SVM and then introduce its variants in the hyperbolic space (Cho et al., 2019; Chien et al., 2021) using l_2 loss and a hinge-loss penalty. Readers may consider using l_1 loss or squared-hinge penalty, and other combinations as well.

ESVM In the classification setting, let $\{((x_i),(y_i))\}_{i=1}^n$ be the training set and $x_i \in \mathbb{R}^n, y_i \in \{1,-1\}, \forall i \in \{1,...,n\}$. To obtain the separating hyperplane, in the soft-margin Euclidean SVM (ESVM), we minimize the margin between the

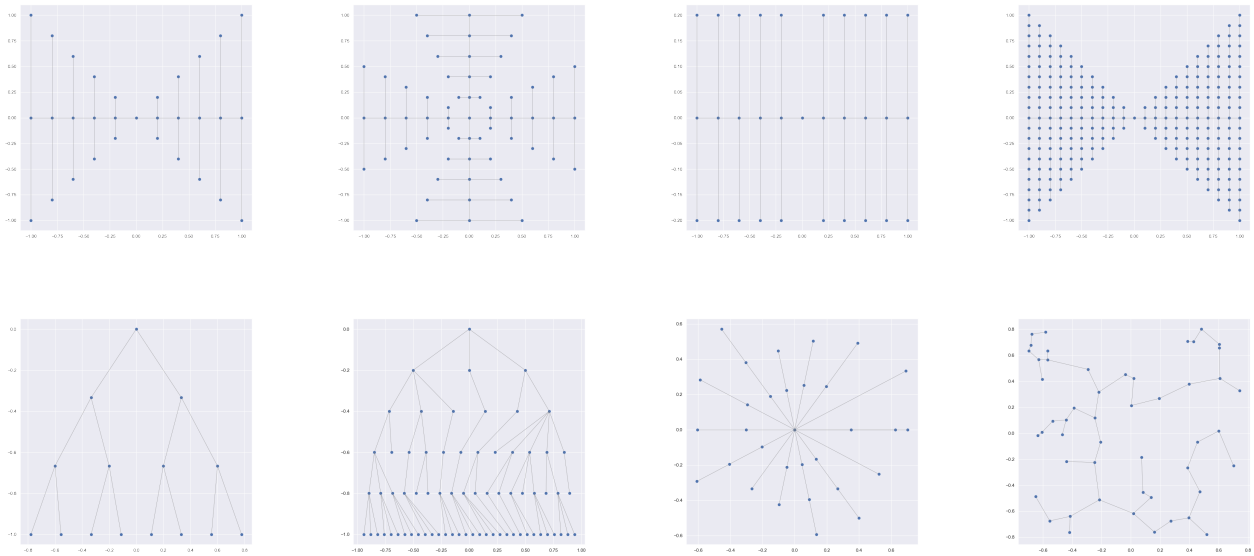

Figure 5. Simulated Trees in $\mathbb{R}^2$. We label figures by Tree 1 - 4 (top row) and Tree 5 - 8 (bottom row). The 2D coordinates of each node are features and pairwise distances computed through shortest path distance on the connected graph.

Table 3. Average distortion δ, max distortion $\delta^{\max}$, and diameter d by Dataset created in Figure 5 and Manifold

tree	manifold	δ	$\delta^{\max}$	d	tree	manifold	δ	$\delta^{\max}$	d
1	raw	1.1954	10.6062	2.8284	5	raw	1.3349	12.7279	1.4142
	$\mathbb{D}^2$	1.0462	4.0546	4.8740		$\mathbb{D}^2$	1.0176	2.0321	4.2523
	$\mathbb{L}^2$	1.0176	2.4511	10.1827		$\mathbb{L}^2$	1.0029	1.2829	7.3850
	$\mathbb{E}^2$	**1.0157**	**2.3158**	**10.9019**		$\mathbb{E}^2$	**1.0019**	**1.1997**	**8.8504**
2	raw	1.1186	14.1421	2.2361	6	raw	1.5080	88.8552	1.4142
	$\mathbb{D}^2$	1.0589	7.1435	4.9757		$\mathbb{D}^2$	1.1243	50.4524	4.3168
	$\mathbb{L}^2$	1.0180	3.2803	10.6719		$\mathbb{L}^2$	1.0392	**21.9948**	8.9204
	$\mathbb{E}^2$	**1.0134**	**2.7408**	**11.4504**		$\mathbb{E}^2$	**1.0269**	22.3743	**9.8190**
3	raw	1.0511	3.0000	2.0396	7	raw	1.2108	17.7132	0.8321
	$\mathbb{D}^2$	1.0321	3.3830	6.3564		$\mathbb{D}^2$	1.2145	8.6454	5.0657
	$\mathbb{L}^2$	1.0190	2.5506	9.6830		$\mathbb{L}^2$	1.0166	3.7655	19.1703
	$\mathbb{E}^2$	**1.0174**	**2.4350**	**10.2328**		$\mathbb{E}^2$	**1.0100**	**1.7825**	**19.8481**
4	raw	1.1539	19.0000	2.8284	8	raw	1.1421	10.7807	0.9220
	$\mathbb{D}^2$	1.0754	14.1074	5.0453		$\mathbb{D}^2$	1.0324	9.7552	4.5670
	$\mathbb{L}^2$	1.0421	8.2933	10.4028		$\mathbb{L}^2$	1.0157	9.2790	8.8324
	$\mathbb{E}^2$	**1.0292**	**6.8504**	**12.7596**		$\mathbb{E}^2$	**1.0148**	**8.4500**	**9.1712**

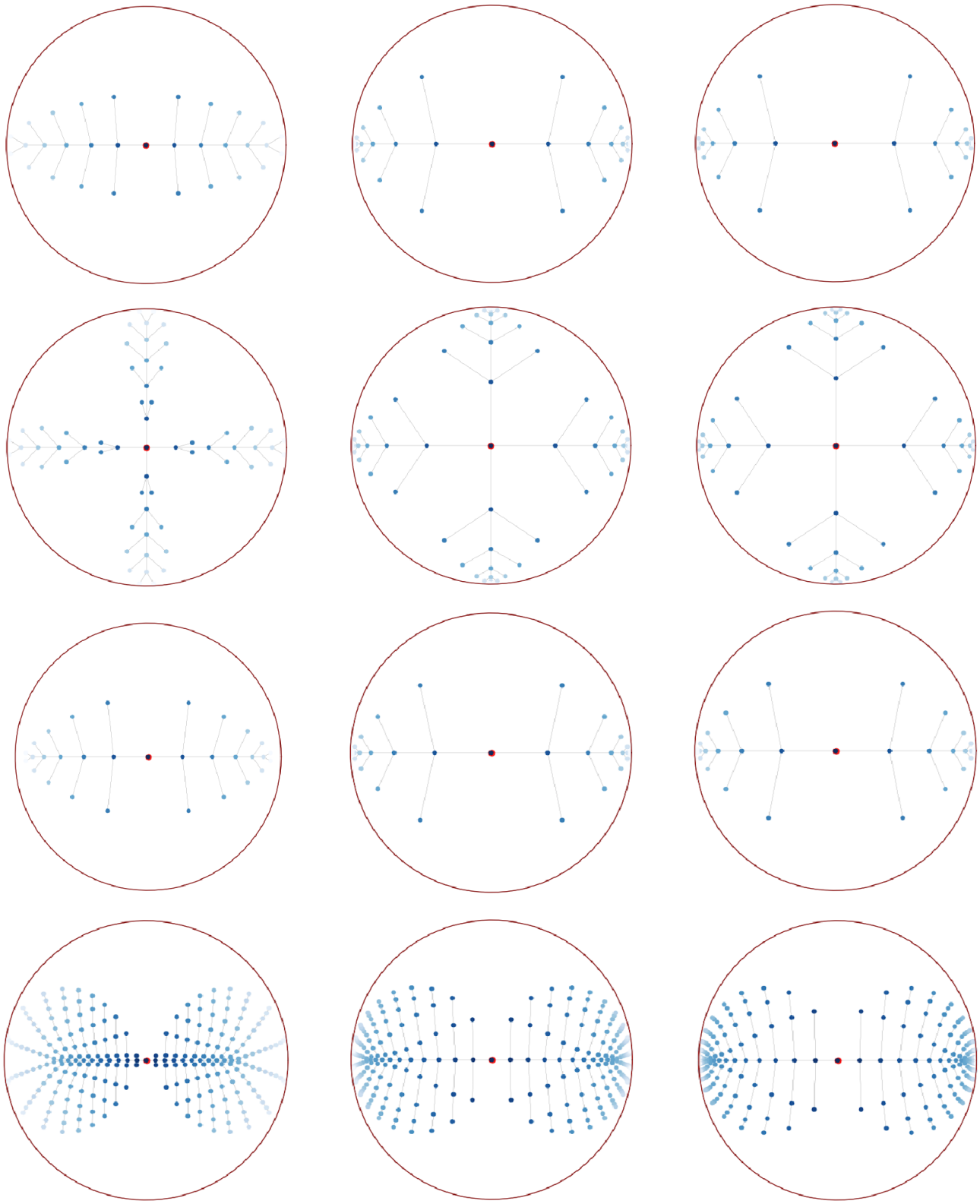

Figure 6. Visualized Embeddings (Tree 1 - 4, from top to bottom row) at the final training epoch (3000). Each column from left to right correspond to Poincaré, Lorentz, and Euclidean parametrization.

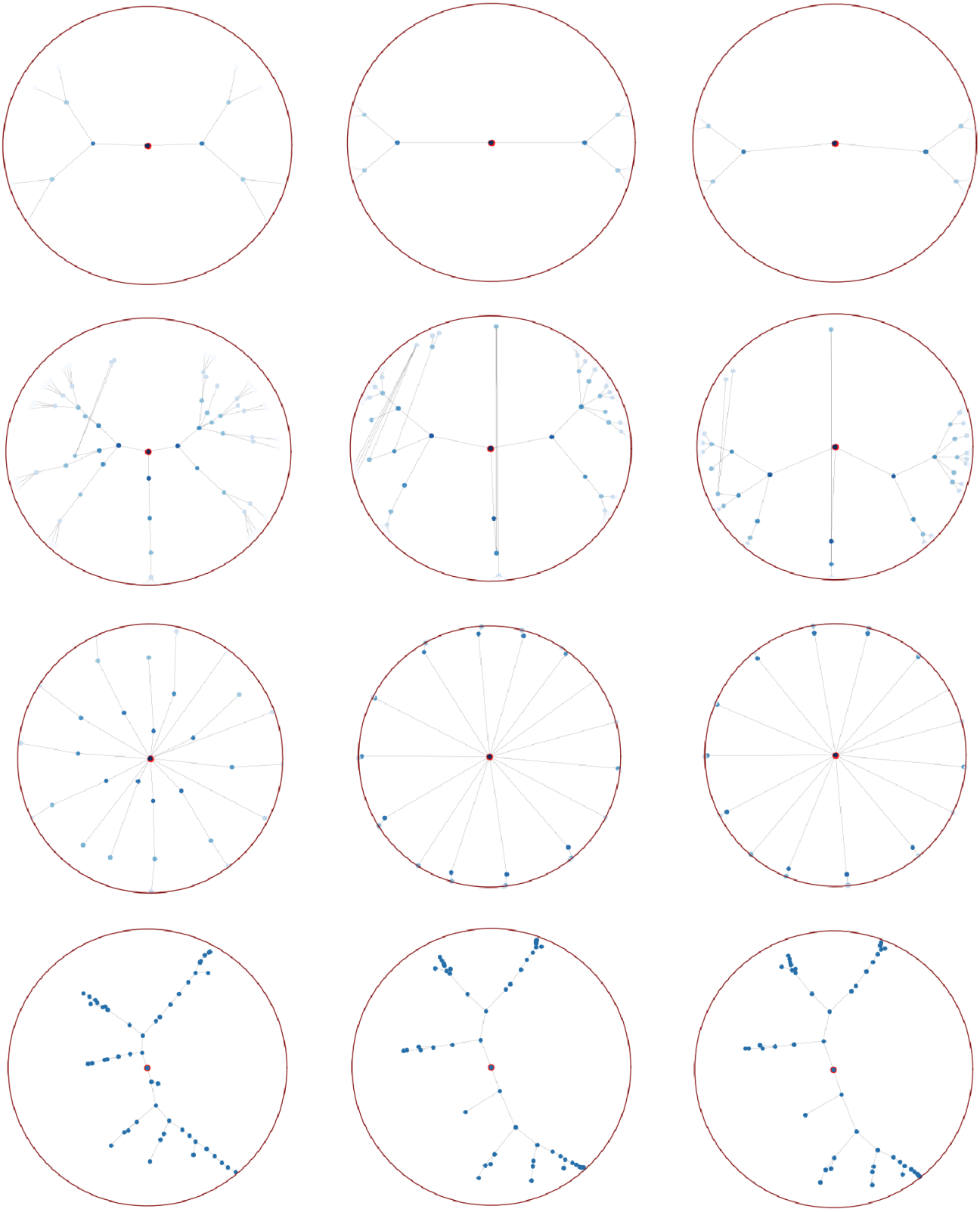

Figure 7. Visualized Embeddings (Tree 5 - 8, from top to bottom row) at the final training epoch (3000). Each column from left to right correspond to Poincaré, Lorentz, and Euclidean parametrization.

hyperplane and the support vectors

$$\min_{w\in\mathbb{R}^n} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} l(y_i\langle w, x_i\rangle)$$

where l is a hinge loss $l(z) = \max(0, 1-z)$, and C controls the tolerance of misclassification.

LSVM Given a hyperbolic classification dataset, i.e., the dataset $\{((x_i, y_i))\}_{i=1}^n$ satisfies that $x_i \in \mathbb{L}^n, y_i \in \{1, -1\}, \forall i \in \{1, ..., n\}$, (Cho et al., 2019) first formulated the soft-margin Lorentz SVM as follows:

$$\begin{aligned}\min_{w\in\mathbb{R}^{n+1}} \frac{1}{2}\|w\|_{\mathbb{L}^n}^2 + C\sum_{i=1}^{n} l_{\mathbb{L}^n}(-y_i[w, x_i])\\ s.t.\ \ [w, w] > 0\end{aligned}$$

where $l_{\mathbb{L}^n}(z) = \max(0, \text{arcsinh}(1) - \text{arcsinh}(z))$. Unfortunately, this problem has a nonconvex constraint and nonconvex objective function.

PSVM Projected gradient descent is slow and may not converge to the global minimum. (Chien et al., 2021) removes the optimization constraints by first precomputing a reference point p on the Poincaré ball and then projecting features via the logarithm map $\log_p$ onto the tangent space $T_p\mathbb{D}^n$, thereby casting hyperbolic SVM back into an ESVM. We call this method PSVM. x However, precomputing a reference point is not part of the SVM heuristics and may not find the maximum margin separating hyperplanes following this approach. The authors use a modified Graham Search (Graham, 1972) to identify convex hulls on the Poincaré ball and use the midpoint of the two convex hulls as the reference point p. This does not have a valid justification when the two convex hulls overlap and may risk introducing a strong bias in the estimated hyperplane.

LSVMPP We build upon the framework of LSVM using our Euclidean parametrization of hyperplanes. Our reformulation lifts the nonconvex constraint $[w, w] > 0$ and does not precompute any reference point. Using our parametrization described in the previous section, we have the following loss:

$$\min_{z\in\mathbb{R}^n, a\in\mathbb{R}} \frac{1}{2}\|z\|^2 + C\sum_{i=1}^{n} l_{\mathbb{L}^n}\left(y_i(\sinh(a)\|z\| x_0 - \cosh(a) z x_r^{\mathrm{T}})\right).$$

This lifts the constraint for optimization, but the objective still remains nonconvex. However, the empirical evaluation shows that this new SVM framework (LSVMPP) still outperforms the previous formulation LSVM, possibly by escaping the local minimums.

C.2. Multiclass Classification Using SVM: Platt Scaling

Suppose now that $y_i \in \{0, 1, ..., k\}$ for all $i \in \{1, ..., n\}$. A common way to train a multiclass classifier is by using the one-vs-all method, in which we train k binary classifiers, using the k-th class as positive samples and negative samples otherwise.

This method only gives us binary predictions if particular samples belong to a certain class, without weighting on the most and least probable class, which ultimately renders the predictions non-interpretable. To solve this issue, Platt scaling (Platt et al., 1999) is a common method to transform binary values in the one-vs-all scheme in multiclass classification to the probabilities of each class. It itself is another machine learning optimization problem. We use ESVM as a demonstration in the following discussion.

In the k-th binary classifier, suppose $w^{(k)}$ is the trained vector determining a hyperplane (intercept included), the decision values are given by

$$f^{(k)}(x_i) = \langle w^{(k)}, x_i\rangle.$$

This serves as a proxy of the distance between a sample to the separating hyperplane. Intuitively, if $f^{(k)}(x_i) > 0$ and if the sample is far away from the hyperplane, we should assign higher confidence, or probability, to the event that x_i belong to the k-th class and lower probability for points closer to the hyperplane. (Platt et al., 1999) argues that a logistic fit works well empirically to measure the associated probability, given by

$$P(y_i = 1|f^{(k)}) = \frac{1}{1 + e^{Af^{(k)}(x_i)+B}}$$

where A, B are parameters trained by minimizing the negative log-likelihood of the training data through some optimization methods, typically through Newton's method.

This procedure repeats for every $f^{(k)}$. That is, we need k (possibly) different sets of (A, B) to transform all binary values into probabilities.

C.2.1. Adaptation of Platt Scaling on Hyperbolic Spaces

Poincaré SVM (PSVM) (Chien et al., 2021) easily adapts to Platt scaling since the hyperplane is trained on the tangent space of a reference point, which is Euclidean, and the decision values are similarly proxies for Euclidean distances (normed distance).

This may become a problem for the other two hyperbolic models on the Lorentz manifold. In (Cho et al., 2019), the authors replace $f^{(k)}$ with

$$f^{(k),\text{LSVM}}(x) = [w^{(k)}, x]$$

where $[\cdot, \cdot]$ denotes the Minkowski product. We argue that this loses its distance proxy functionality, and could be made more of a certain distance metric by

$$f^{(k),\text{LSVMPP}}(x) = \text{arcsinh}\left(-\frac{[w^{(k)}, x]}{\left\|w^{(k)}\right\|_{\mathbb{L}^n}}\right)\left\|w^{(k)}\right\|_{\mathbb{L}^n}$$

and the last formulation is the input for the hyperbolic Platt scaling. Note that $\left|\text{arcsinh}\left(-\frac{[w^{(k)},x]}{\left\|w^{(k)}\right\|_{\mathbb{L}^n}}\right)\right|$ corresponds to the distance between the point x and the hyperplane determined by $w^{(k)}$ (see (Cho et al., 2019, Lemma 4.1)[4]). So the above formula denotes a normed signed distance between the point x and the hyperplane determined by $w^{(k)}$. We report that empirically there is an edge in accuracy using this formulation instead of the previous one (see Table 5). However, it remains to show whether the logistic fit is still a valid assumption, particularly for datasets with a latent hierarchical structure on hyperbolic spaces.

C.3. Hyperbolic SVM Implementation and Hyperparameters

We rely on scikit-learn (Pedregosa et al., 2011) for the realization of Euclidean and Poincaré SVM. The relevance of Poincaré SVM training is that after identifying a reference point and projecting Poincaré features onto its tangent space, features are Euclidean and hence Euclidean SVM training applies, although its intercept should be set to 0, since the precomputed reference point already encodes the intercept information. For Lorentz SVM, we inherit from the code provided in (Cho et al., 2019; Chien et al., 2021) and modify it for better run-time performances; and for our model LSVMPP, we use PyTorch (Paszke et al., 2017) to help to compute the gradient automatically.

Hyperparameters for four different SVM models should be different since they have different magnitudes in both loss and penalty terms. Margins, in particular, in the Euclidean, Poincaré, and Lorentz spaces for the same set of data are different. Therefore, hyperparameter search should be made separate for each model.

The best performances of the Euclidean and Poincaré SVM are both using $C = 5$, with a learning rate of 0.001 and 3000 epochs. An early stop mechanism is built in the scikit-learn implementation (Pedregosa et al., 2011) of Euclidean SVM, so not all 3000 epochs are utilized, and empirically 500 epochs suffice the respective optimality for all tested data; the best performance of LSVM and LSVMPP are in general brought by $C = 0.5$, with a learning rate around 10^{-10} (depending on the initial scale of the dataset) with 500 epochs. We note that the abnormally small learning rate is explained by the necessity of escaping the local minimum of a complex non-convex objective function.

C.4. Details of Synthetic Data Generation

This section provides more detail on generation of synthetic datasets. As noted previously, we have two types of synthetic datasets: Gaussian mixtures and explicit trees. Gaussian mixture datasets are named sim_data_1, sim_data_2, and sim_data_3 and explicit trees are named sim_data_4, sim_data_5, sim_data_6. All synthetic datasets are visualized in Figure 8.

In particular, to generate an explicit tree dataset, we simulate a birth-and-death process. In this process, the distribution of

[4]There is a slight imprecision in (Cho et al., 2019, Lemma 4.1) that the absolute value sign was missing.

offspring follows a Poisson distribution with a parameter value of $\lambda = 3$, with an additional 1 added (resulting in a Poisson random variable starting with 1). The three plotted explicit trees represent three different realizations of this process. Once the tree structure is established, we initialize Euclidean embeddings that closely resemble the final embeddings in Figure 8. We then apply the tree embedding method, as described in Section 5.1, utilizing Euclidean parametrization to optimize the embedding. We train with SGD using a learning rate of 10 for 50000 epochs.

C.5. Full Results

We visualize all datasets on the Poincaré ball in Figure 8.

The average accuracy and F1 scores of five different runs on the same train-test split of each dataset are reported below in Table 4 and Table 5.

To maintain consistency in the evaluation, we adopt the train-test split provided in the GitHub referenced in (Chien et al., 2021) for the olsson, CIFAR, and fashion-MNIST datasets. For all other datasets, we utilize a 75%/25% train-test split stratified based on the class assignments.

In addition, we also compare to a model using Platt scaling on raw output, named "LSVMPP (raw)" in the table below. "LSVMPP (arcsinh)" is the version with adapted input for Platt scaling. The two versions share the same learning rate. See Appendix C.2.1 for more details. Notice that LSVMPP (raw) performs better than the adapted version when Euclidean SVM is at the top-notch; otherwise, it is worse and less stable in performance than the adapted version, indicating a lesser fit of the Platt training step with respect to the raw decision values.

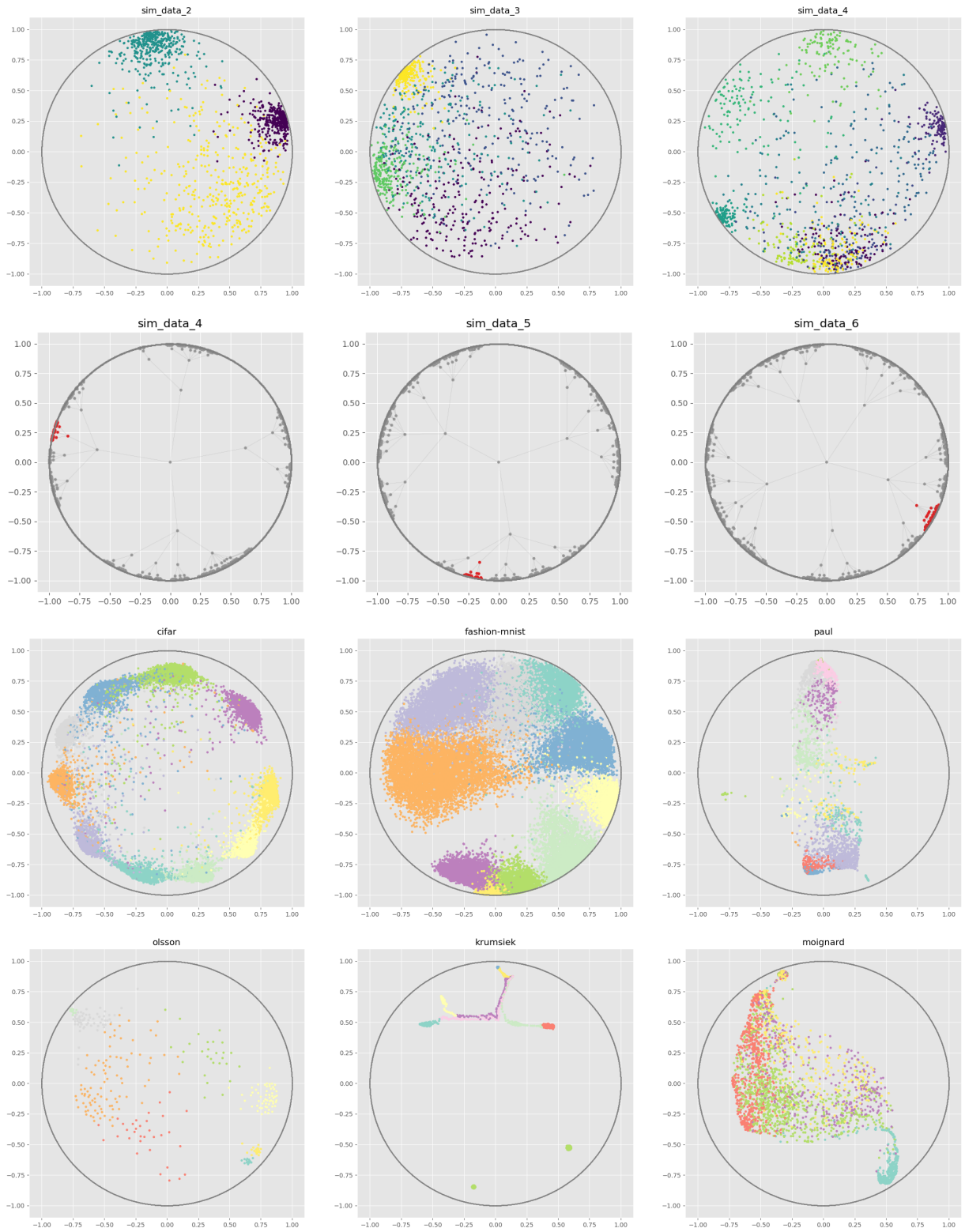

Figure 8. Simulated (first two rows, with $k = 3, k = 5, k = 10$ from left to right for the first row respectively and $k = 2$ for the second row) and real (last two rows) datasets: CIFAR ($k = 10$), fashion-MNIST ($k = 10$), paul ($k = 19$), olsson ($k = 8$), krumsiek ($k = 11$), and moignard $k = 7$. In each plot, different colors represent different classes. For the explicit tree datasets we also include the tree edges.

Table 4. Mean accuracy and macro F1 score of synthetic Dataset in Figure 8.

	algorithm	accuracy (%)	F1
sim 1	ESVM	94.67 ± 0.00	0.9465 ± 0.00
	LSVM	94.67 ± 0.00	0.9465 ± 0.00
	PSVM	91.66 ± 0.00	0.9151 ± 0.00
	LSVMPP (raw)	94.67 ± 0.00	0.9464 ± 0.00
	LSVMPP (arcsinh)	94.67 ± 0.00	0.9464 ± 0.00
sim 2	ESVM	68.67 ± 0.00	0.6656 ± 0.00
	LSVM	66.66 ± 0.00	0.6336 ± 0.00
	PSVM	64.66 ± 0.00	0.6143 ± 0.00
	LSVMPP (raw)	$\mathbf{70.33 \pm 0.00}$	$\mathbf{0.6920 \pm 0.00}$
	LSVMPP (arcsinh)	69.00 ± 0.00	0.6840 ± 0.00
sim 3	ESVM	61.67 ± 0.00	$\mathbf{0.5908 \pm 0.00}$
	LSVM	39.00 ± 0.00	0.3236 ± 0.00
	PSVM	61.00 ± 0.00	0.5606 ± 0.00
	LSVMPP (raw)	60.67 ± 0.00	0.5468 ± 0.00
	LSVMPP (arcsinh)	$\mathbf{61.93 \pm 0.01}$	0.5881 ± 0.01
sim 4	ESVM	93.40 ± 0.00	0.0000 ± 0.00
	LSVM	96.04 ± 3.24	0.4000 ± 0.49
	PSVM	99.53 ± 0.00	0.9630 ± 0.00
	LSVMPP (raw)	$\mathbf{100.00 \pm 0.00}$	$\mathbf{1.0000 \pm 0.00}$
	LSVMPP (arcsinh)	$\mathbf{100.00 \pm 0.00}$	$\mathbf{1.0000 \pm 0.00}$
sim 5	ESVM	96.25 ± 0.00	0.0000 ± 0.00
	LSVM	96.25 ± 0.00	0.0000 ± 0.00
	PSVM	99.63 ± 0.00	0.9474 ± 0.00
	LSVMPP (raw)	99.63 ± 0.00	0.9474 ± 0.00
	LSVMPP (arcsinh)	99.63 ± 0.00	0.9474 ± 0.00
sim 6	ESVM	97.64 ± 0.00	0.7805 ± 0.00
	LSVM	99.63 ± 0.13	0.9711 ± 0.01
	PSVM	97.91 ± 0.00	0.8095 ± 0.00
	LSVMPP (raw)	99.53 ± 0.42	0.9648 ± 0.03
	LSVMPP (arcsinh)	$\mathbf{99.63 \pm 0.13}$	$\mathbf{0.9714 \pm 0.01}$

Table 5. Mean accuracy and macro F1 score of real Dataset in Figure 8.

	algorithm	accuracy (%)	F1
CIFAR-10	ESVM	91.88 ± 0.00	0.9191 ± 0.00
	LSVM	91.88 ± 0.00	0.9189 ± 0.00
	PSVM	91.81 ± 0.00	0.9182 ± 0.00
	LSVMPP (raw)	91.94 ± 0.00	0.9195 ± 0.00
	LSVMPP (arcsinh)	$\mathbf{91.96 \pm 0.00}$	$\mathbf{0.9197 \pm 0.00}$
fashion-MNIST	ESVM	86.37 ± 0.00	0.8665 ± 0.00
	LSVM	71.59 ± 0.07	0.6588 ± 0.08
	PSVM	86.57 ± 0.00	0.8665 ± 0.00
	LSVMPP (raw)	89.35 ± 0.00	0.8939 ± 0.00
	LSVMPP (arcsinh)	$\mathbf{89.49 \pm 0.00}$	$\mathbf{0.8955 \pm 0.00}$
paul	ESVM	55.05 ± 0.00	0.4073 ± 0.00
	LSVM	58.36 ± 0.07	0.4517 ± 0.00
	PSVM	55.25 ± 0.00	0.3802 ± 0.00
	LSVMPP (raw)	62.55 ± 0.14	0.4579 ± 0.01
	LSVMPP (arcsinh)	$\mathbf{62.64 \pm 0.05}$	$\mathbf{0.5024 \pm 0.00}$
olsson	ESVM	72.72 ± 0.00	0.4922 ± 0.00
	LSVM	81.82 ± 0.00	0.7542 ± 0.00
	PSVM	$\mathbf{88.63 \pm 0.00}$	$\mathbf{0.8793 \pm 0.00}$
	LSVMPP (raw)	80.91 ± 0.45	0.7142 ± 0.03
	LSVMPP (arcsinh)	84.09 ± 0.00	0.8429 ± 0.00
krumsiek	ESVM	82.19 ± 0.00	0.6770 ± 0.00
	LSVM	85.62 ± 0.39	0.6933 ± 0.92
	PSVM	84.06 ± 0.00	0.6908 ± 0.00
	LSVMPP (raw)	83.75 ± 0.28	0.6403 ± 0.01
	LSVMPP (arcsinh)	$\mathbf{86.25 \pm 0.00}$	$\mathbf{0.7079 \pm 0.00}$
moignard	ESVM	$\mathbf{67.78 \pm 0.00}$	$\mathbf{0.5934 \pm 0.00}$
	LSVM	64.88 ± 0.38	0.5502 ± 0.22
	PSVM	60.77 ± 0.00	0.5167 ± 0.00
	LSVMPP (raw)	65.77 ± 0.13	0.5671 ± 0.00
	LSVMPP (arcsinh)	65.22 ± 0.63	0.5719 ± 0.02