MSc Artificial Intelligence
Master Thesis

Lorentzian Neural Networks++

by
Ricardo Chávez Torres
15239470

November 12, 2025

36 ECTS
February 2025 - July 2025

Supervisors:
MSc. Max van Spengler

Examiner:
Prof. Pascal Mettes

Second reader:
MSc. Max van Spengler

Universiteit van Amsterdam

Abstract

In recent years, there has been a growing interest in methods that embed neural network representations or parameters in hyperbolic manifolds due to their advantages in representing hierarchical data and allowing for more compact models and embeddings.

To perform computations in hyperbolic space, we use the so-called models of hyperbolic space, all of which are isometric, meaning that we can transform between them without altering distances. The most common choices are the Poincaré ball model and the Lorentz model. Many deep learning operations have been redefined in these different hyperbolic models (Chen et al., 2022; Ganea et al., 2018; Shimizu et al., 2021). Still, their geometric definitions differ, resulting in two distinct formulations of hyperbolic layers: the distance to hyperplanes in Poincaré, and the projection to a hyperboloid in Lorentz. This makes it unclear whether performance differences arise from distinct geometric definitions or the different numerical representation capabilities of the models. As a result, the choice between models is somewhat arbitrary.

This thesis takes a step forward by introducing a layer that combines the strengths of both frameworks, retaining the computational efficiency of Lorentz models while preserving the geometric rigor of the Poincaré formulation. Our main contributions are:

Lorentzian Fully Connected Layers: We derive the geometric formulation of the Poincaré fully connected layers based on the distance to hyperplanes in the Lorentz Hyperbolic Manifold.

Benchmark against original Poincaré and Lorentz models: We perform a thorough comparison against hyperbolic models built using Poincaré and Lorentz layers. We demonstrate a speedup of 33% compared to Poincaré networks when relaxing the conditions to allow more flexible activation functions.

Accelerating inference time: We propose caching some intermediate values during inference to avoid recomputing them repeatedly and accelerating inference enormously without compromising model size.

With these contributions we aim to advance methods for doing fully hyperbolic deep learning.

Keywords: Hyperbolic Deep Learning, Representation Learning.

Contents

List of Figures

List of Tables

Introduction 1

Representation learning stands as a cornerstone of modern machine learning, focusing on the discovery of effective data representations, or embeddings, that simplify downstream tasks. The vast majority of deep learning models, from convolutional neural networks to transformers, implicitly assume that this underlying latent space possesses a Euclidean geometry. This assumption, while powerful and effective for many data types, is not universally optimal. In recent years, a growing field of research known as hyperbolic deep learning has emerged to challenge this convention, proposing that certain types of data are more naturally modeled in curved, non-Euclidean spaces.

1.1 Why Hyperbolic Space?

Among the various non-Euclidean geometries, hyperbolic space has emerged as a particularly powerful candidate for modeling hierarchical data (Mettes et al., 2024; Peng et al., 2022). As a manifold with constant negative curvature, its geometry exhibits properties that are remarkably well-suited for representing trees and graphs. Unlike Euclidean space, where the circumference and volume of a circle grow polynomially with its radius, in hyperbolic space, they grow exponentially. This exponential expansion of space allows for the embedding of large, complex tree-like structures with significantly lower distortion and in far fewer dimensions than would be required in Euclidean space (Yang et al., 2022).

This geometric alignment between hyperbolic space and hierarchical data is not merely a theoretical curiosity; it offers tangible benefits. By embedding data into hyperbolic manifolds, we can create more compact, parsimonious, and powerful models. The inductive bias provided by the geometry allows the model to learn more meaningful representations, capturing the notion of hierarchy directly within the embedding space. This has led to groundbreaking improvements in tasks such as graph embedding and link prediction (Chami et al., 2019).

1.2 The Problem: Choosing Which Hyperbolic Model

To perform computations and build neural networks fully in hyperbolic space, one must work in a specific model of hyperbolic space. While all models of hyperbolic space are isometric—meaning distances and angles are preserved when mapping between them—their coordinates and mathematical formulations differ significantly. The two most prevalent models in deep learning are the Poincaré ball model and the Lorentz model.

This has led to a split in the field: foundational work has developed neural operations independently within each model. In the Poincaré ball, layers like fully connected networks are often defined using distances to hyperplanes. In contrast, the Lorentz model typically relies on projections onto the hyperboloid. These differing geometric foundations have resulted in two parallel but largely isolated ecosystems of hyperbolic neural layers.

This divergence raises a fundamental question: which model should we use? Performance differences between Poincaré and Lorentz based networks are not well understood—it's unclear whether they stem from geometric definitions or from the numerical properties of the models themselves. As a result, model choice is often driven by convention or implementation convenience, rather than by a principled comparison. The lack of a unified framework hinders systematic evaluation, integration, and advancement across these approaches.

Each method comes with its own set of tradeoffs. The Poincaré model stands out for its robust, geometrically principled formulation—it provides a mathematically sound analogue to the Euclidean linear layer. However, this rigor comes at a high computational cost, making it significantly slower than both Euclidean and Lorentz-based approaches. This often tips the balance in favor of the Lorentz model.

The Lorentz model, in contrast, is computationally efficient but lacks a similarly principled geometric foundation. By relaxing key constraints of hyperbolic transformations, it sacrifices some of the intrinsic benefits of hyperbolic geometry—effectively reducing the gain to just concatenating an additional feature, essentially doing feature engineering.

In this work, we address that gap by introducing a geometrically principled formulation within the Lorentz model, equivalent in rigor to that of the Poincaré approach. Our method achieves significantly faster computation, even outperforming biased Euclidean layers in certain inference scenarios.

1.3 Research Questions and Objectives

This thesis aims to bridge the gap between the Poincaré and Lorentz formulations of hyperbolic deep learning. To address the ambiguity outlined above, we formalize our investigation through the following research questions:

- Can the geometric formulation of the Poincaré fully connected layer, based on distances to hyperplanes, be mathematically derived and expressed within the Lorentz model to create a unified framework?
- Can this unified formulation serve as a foundation for building a complete suite of deep learning operations—including convolutions, attention, and activations—entirely within the Lorentz model?
- How do fully Lorentzian neural networks, built upon this unified framework, perform against the original Poincaré and Lorentz-based models in terms of computational speed, accuracy, and memory efficiency?

- Can the inference process of these networks be further accelerated by exploiting the properties of the Lorentz model to reduce redundant computations?

By answering these questions, this thesis seeks to provide a clear and direct comparison between the two dominant models of hyperbolic space, moving the field from arbitrary choices to principled design.

1.4 Contributions

To achieve these objectives, this thesis makes the following primary contributions:

- **A Unified Hyperbolic Layer Formulation:** We present a formal mathematical derivation that translates the hyperplane-based fully connected layer of the Poincaré model into the Lorentz model. This creates a single, unified layer that serves as a common ground for direct comparison.
- **Benchmarking and Analysis:** We conduct a thorough empirical evaluation, comparing our unified Lorentzian networks against established Poincaré and Lorentz baselines. Our results demonstrate a significant speedup of 33% over Poincaré-based networks, particularly when leveraging the flexibility of our proposed activation functions, without sacrificing performance.
- **Accelerating the inference:** We introduce a caching strategy for intermediate values that are unique to the inference pass of our Lorentzian networks. This method avoids repeated calculations and further accelerates model execution time in inference without compromising model size.

1.5 Thesis Outline

The remainder of this thesis is structured as follows. **Chapter 2** provides the necessary mathematical background on Riemannian and hyperbolic geometry. **Chapter 3** reviews related work in fully hyperbolic deep learning, contextualizing our contributions within the current literature. **Chapter 4** presents our primary theoretical contribution: the detailed derivation of the geometrically grounded Lorentzian fully connected layer and its associated operations. **Chapter 5** describes our experimental setup and presents the results of our benchmarking and classification tasks. Finally, **Chapter 6** discusses the broader implications of our findings, acknowledges limitations, and outlines promising directions for future research.

Mathematical Preliminaries 2

In this chapter, we will first introduce the mathematical preliminaries to hyperbolic geometry, necessary to understand hyperbolic deep learning methods. A reader that is familiar with hyperbolic geometry and more specifically, with the concept of Poincaré ball and Lorentz hyperboloid hyperbolic models, how to measure distances between points in the models, geodesic arc, exponential and logarithmic maps can skip this chapter.

2.1 Foundations from Differential Geometry

Before delving into the specifics of hyperbolic geometry, we introduce some fundamental concepts from differential geometry that provide the language to describe these spaces formally.

2.1.1 Manifolds and Tangent Spaces

A **manifold** is a topological space that, on a local scale, can be approximated by an Euclidean space. An n-dimensional manifold, $\mathcal{M}$, is a space where every point has a neighborhood that is topologically equivalent (homeomorphic) to an open subset of $\mathbb{R}^n$.

At each point p on a manifold $\mathcal{M}$, we can define a **tangent space**, denoted $T_p\mathcal{M}$. The tangent space is a vector space that consists of all possible "velocities" or directional derivatives at that point. It can be visualized as the hyperplane tangent to the manifold at p.

2.1.2 Riemannian Manifolds and Metrics

A **Riemannian manifold** $(\mathcal{M}, \mathfrak{g})$ is a smooth manifold $\mathcal{M}$ equipped with a **Riemannian metric** $\mathfrak{g}$.

The Riemannian metric $\mathfrak{g}$ equips the tangent space at each point with an inner product that varies smoothly when varying the point.

$$\mathfrak{g}_p : T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R} \tag{2.1}$$

This metric allows us to define geometric notions such as the length of curves, angles between tangent vectors, and volume.

One example of a Riemannian manifold is the Euclidean space $\mathbb{R}^n$ equipped with the standard metric. Here, the manifold $\mathcal{M} = \mathbb{R}^n$ and the Riemannian metric $\mathfrak{g}$ is given by the standard dot product on $\mathbb{R}^n$:

$$\mathfrak{g}_p(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v} \quad \forall\, \mathbf{u}, \mathbf{v} \in T_p\mathbb{R}^n. \tag{2.2}$$

2.1.3 Geodesics

A **geodesic** is the generalization of a straight line to a Riemannian manifold. It is a curve that represents the shortest path between two points. The length of a geodesic between two points gives the distance between them on the manifold.

2.1.4 Learning on Riemannian Manifolds

Optimizing a loss function $L(\boldsymbol{\theta})$ for parameters $\boldsymbol{\theta}$ on a Riemannian manifold $\mathcal{M}$ requires generalizing standard gradient descent. The Euclidean update rule, $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t)$, is ill-defined because one cannot subtract a tangent vector from a point on the manifold.

The generalization, **Riemannian gradient descent**, adapts the optimization process to the manifold's geometry. At each step, it first computes the **Riemannian gradient**, $\nabla_{\mathcal{M}} L(\boldsymbol{\theta}_t)$, which is a vector in the tangent space $T_{\boldsymbol{\theta}_t}\mathcal{M}$. Then, it updates the parameters by moving along a geodesic. This is achieved using the **exponential map**, which takes the tangent vector and maps it to a new point on the manifold. The update rule is:

$$\boldsymbol{\theta}_{t+1} = \exp_{\boldsymbol{\theta}_t}(-\eta \nabla_{\mathcal{M}} L(\boldsymbol{\theta}_t)) \tag{2.3}$$

Essentially, the optimization step is performed in the local tangent space (a vector space) and then projected back onto the manifold.

2.2 Hyperbolic Geometry

Hyperbolic geometry is a non-Euclidean geometry where Euclid's fifth postulate, the parallel postulate, is not true. In contrast to Euclidean geometry, where for a given line and an external point there exists exactly one unique parallel line passing through that point, in hyperbolic geometry there are infinitely many such parallel lines. This concept is illustrated in Figure 2.1.

There exist many isometric analytical models that allow us to study hyperbolic geometry, here we will review the Lorentz model also called the Hyperboloid model, and the Poincaré model also called the Ball model or interior disk model.

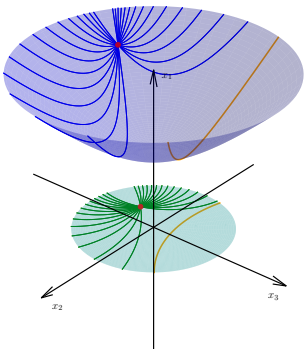


Figure 2.1: A line with multiple parallels that pass through the same point in hyperbolic space. Blue is the Lorentz Hyperboloid model and green is the Poincaré Ball model.

2.2.1 The Lorentz Inner Product Space: $(\mathbb{R}^n, \circ)$

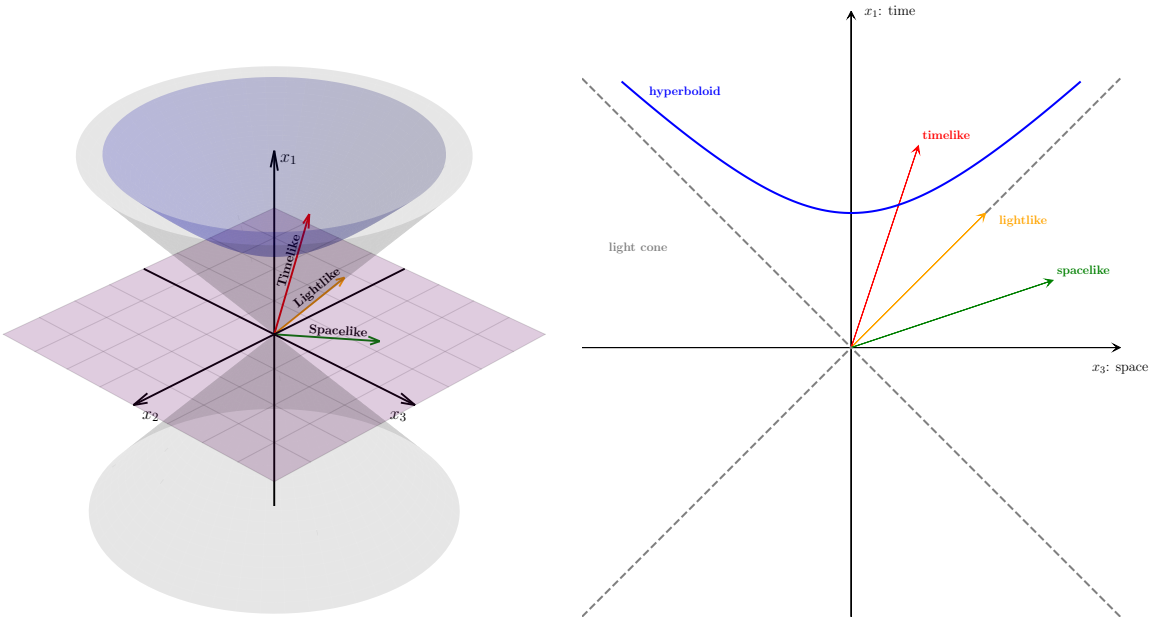
Before defining the Lorentz model, we first define the ambient space in which it lives.

Let $\mathbf{x}$ and $\mathbf{y}$ be vectors in $\mathbb{R}^n$. We define the **Lorentzian inner product** [1] between $\mathbf{x}$ and $\mathbf{y}$ as:

$$\mathbf{x} \circ \mathbf{y} = -x_1 y_1 + x_2 y_2 + \ldots + x_n y_n = -x_1 y_1 + \overline{\mathbf{x}} \cdot \overline{\mathbf{y}} \tag{2.4}$$

Where $\overline{\mathbf{x}} = (x_2, \ldots, x_n)$ are called the spatial components of $\mathbf{x}$ and x_1 its time component.

1: Notice the **Lorentzian inner product** is not an inner product under the usual definition because it is not positive definite. It is instead a non-degenerate symmetric bilinear form or pseudo-inner product. But we will keep this notation to match Ratcliffe, 2019.



(a) 3D Lorentz product space.

(b) 2D section of 3D Lorentz product space.

Figure 2.2: An illustration of the hyperboloid and classification of vectors in the 3D Lorentz product space. In **(a)** we see the space hyperplane in purple, the light cone for which every point $\mathbf{x}$ has $\|\mathbf{x}\|_{\mathcal{L}} = 0$, the hyperboloid in blue, and three vectors in red, yellow, and green. **(b)** is a 2D section of the former.

The **Lorentzian norm**[2] is defined as:

$$\|\mathbf{x}\|_{\mathcal{L}} = (\mathbf{x} \circ \mathbf{x})^{1/2} \tag{2.5}$$

2: Similarly, note that it is not a norm but instead a pseudo-norm as it is not positive definite and does not follow the triangle inequality

Observe that the Lorentzian norm of a vector can be zero for non-zero vectors and can yield imaginary values. We will call the **magnitude of a vector** to the magnitude of the Lorentzian norm and denote it with triple vertical lines $\|\|\mathbf{x}\|\|_{\mathcal{L}}$. The magnitude of a vector takes real values.

The pair composed by $\mathbb{R}^n$ equipped with the Lorentzian inner product $\circ$ is the n-**Lorentz product space** $(\mathbb{R}^n, \circ)$.

A vector $\mathbf{x}$ in $\mathbb{R}^n$ is classified as:

light-like if $\|\mathbf{x}\|^2_{\mathcal{L}} = 0$. These vectors lie on a cone equidistant in euclidean distance to the space hyperplane and the time axis.
time-like if $\|\mathbf{x}\|^2_{\mathcal{L}} < 0$. These vectors are closer in Euclidean distance to the time axis.
space-like if $\|\mathbf{x}\|^2_{\mathcal{L}} > 0$. These vectors are closer in Euclidean distance to the space hyperplane.

The **parity** of a vector is the sign of its time element, **positive** if $x_1 > 0$ and **negative** if $x_1 < 0$.

A vector subspace $V \subset \mathbb{R}^n$ is classified as:

time-like if V contains a time-like vector.
space-like if every non-zero vector in V is space-like.
light-like otherwise.

The **Lorentzian complement** of a vector subspace $V \subset \mathbb{R}^n$ is defined as:

$$V^L = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} \circ \mathbf{y} = 0 \ \forall \mathbf{y} \in V\}. \tag{2.6}$$

V^L is also a subspace of $\mathbb{R}^n$ and has the following properties: $\dim V + \dim V^L = n$ (if V is timelike or spacelike), $(V^L)^L = V$ and, V is time-like (space-like) if and only if V^L is space-like (time-like).

The **time-like angle** between two time-like vectors $\mathbf{x}$ and $\mathbf{y}$ with the same parity is $\eta(\mathbf{x}, \mathbf{y})$, defined by:

$$\mathbf{x} \circ \mathbf{y} = \|\mathbf{x}\|_{\mathscr{L}} \|\mathbf{y}\|_{\mathscr{L}} \cosh \eta(\mathbf{x}, \mathbf{y}). \tag{2.7}$$

$\eta(\mathbf{x}, \mathbf{y})$ is unique and non-negative.

Notice that the time-like angle follows a definition very similar to that of the euclidean angle of two vectors[3]. This parallelism will follow through many concepts and will allow us to understand better some definitions in hyperbolic space.

3: Let $\mathbf{x}$, $\mathbf{y}$ be two vectors in a Euclidean space. The Euclidean angle $\eta_E(\mathbf{x}, \mathbf{y})$ between them is defined via the identity:

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\|_E \|\mathbf{y}\|_E \cos \eta_E(\mathbf{x}, \mathbf{y})$$

2.2.2 The Lorentz Hyperboloid Model: $\mathbb{L}^n_\kappa$

The **Lorentz hyperboloid model** $\mathbb{L}^n_\kappa$ of n-dimensional hyperbolic space with constant negative curvature $-\kappa$ ($\kappa > 0$) is defined as a submanifold of the Lorentz inner product space:

$$\mathbb{L}^n_\kappa = \left\{ \mathbf{z} \in \mathbb{R}^{n+1} \mid \|\mathbf{z}\|^2_{\mathscr{L}} = -\frac{1}{\kappa} \text{ and } z_1 > 0 \right\}. \tag{2.8}$$

The Lorentz model $\mathbb{L}^n_\kappa$ is a **Riemannian manifold**. Its Riemannian metric is the one induced on its tangent spaces by the ambient Lorentzian inner product on $\mathbb{R}^{n+1}$.

The condition $z_1 > 0$ restricts the model to one sheet of the hyperboloid. This definition is analogous to that of a sphere, where points are equidistant to the center, with the squared Euclidean norm $\|\mathbf{z}\|^2_E = r^2$. For the hyperboloid, the constant $-1/\kappa$ plays a similar role to r^2 for a sphere. This prompts us to define the radius of the hyperboloid as $r = \sqrt{-1/\kappa}$. With this perspective, the hyperboloid could be viewed as the sphere of imaginary radius under the Lorentz metric[4].

4: Although this does not account for the negative sheet of the hyperboloid

All vectors in $\mathbb{L}^n_\kappa$ are positive and time-like, meaning that their Lorentzian norm squared is negative.

Note that the norm $\|\mathbf{v}\|_{\mathscr{L}}$ for a non zero tangent vector $\mathbf{v}$ is positive, as all tangent spaces are space-like.

We call $\mathbf{0}_{\mathbb{L}^n_\kappa} = (1/\sqrt{\kappa}, \underbrace{0, \ldots, 0}_{n})$ the origin or the zero.

Hyperbolic Distance

The distance between two points $\mathbf{x}, \mathbf{y} \in \mathbb{L}^n_\kappa$ is the length of the geodesic connecting them. It is given by:

$$d_{\mathbb{L}}(\mathbf{x}, \mathbf{y}) = \frac{1}{\sqrt{\kappa}} \operatorname{arccosh}\left(-\kappa(\mathbf{x} \circ \mathbf{y})\right). \tag{2.9}$$

Notice this is the magnitude of the radius $\|r\| = \|\sqrt{-1/\kappa}\|$ times the angle η, a similar definition as the spherical distance of two points in a sphere.

When computing the distance to the origin $\mathbf{x} = \mathbf{0}_{\mathbb{L}_\kappa^n}$ the formula simplifies to:

$$d_{\mathbb{L}}(\mathbf{0}_{\mathbb{L}_\kappa^n}, \mathbf{y}) = \frac{1}{\sqrt{\kappa}} \operatorname{arccosh}\left(\sqrt{\kappa} y_1\right). \tag{2.10}$$

Exponential Map

The **exponential map** $\exp_{\mathbf{x}} : T_{\mathbf{x}}\mathbb{L}_\kappa^n \to \mathbb{L}_\kappa^n$ traces a geodesic starting at $\mathbf{x}$ in the direction of a tangent vector $\mathbf{v} \in T_{\mathbf{x}}\mathbb{L}_\kappa^n$ for a distance equal to the magnitude of $\mathbf{v}$. It is given by:

$$\exp_{\mathbf{x}}(\mathbf{v}) = \cosh(\sqrt{\kappa}\,\|\mathbf{v}\|_{\mathcal{L}})\mathbf{x} + \sinh(\sqrt{\kappa}\,\|\mathbf{v}\|_{\mathcal{L}})\frac{\mathbf{v}}{\sqrt{\kappa}\,\|\mathbf{v}\|_{\mathcal{L}}}. \tag{2.11}$$

The exponential map from the origin simplifies to:

$$\exp_{0_{\mathbb{L}_\kappa^n}}(\mathbf{v}) = \left(\frac{1}{\sqrt{\kappa}}\cosh(\sqrt{\kappa}\,\|\overline{\mathbf{v}}\|_E), \quad \sinh(\sqrt{\kappa}\,\|\overline{\mathbf{v}}\|_E)\frac{\overline{\mathbf{v}}}{\sqrt{\kappa}\,\|\overline{\mathbf{v}}\|_E}\right) \tag{2.12}$$

Logarithmic Map

The **logarithmic map** $\log_{\mathbf{x}} : \mathbb{L}_\kappa^n \to T_{\mathbf{x}}\mathbb{L}_\kappa^n$ is the inverse of the exponential map. It maps a point $\mathbf{y} \in \mathbb{L}_\kappa^n$ to the tangent vector $\mathbf{v} \in T_{\mathbf{x}}\mathbb{L}_\kappa^n$ at $\mathbf{x}$ that corresponds to the geodesic connecting $\mathbf{x}$ and $\mathbf{y}$. The formula is:

$$\log_{\mathbf{x}}(\mathbf{y}) = \frac{\operatorname{arccosh}(-\kappa(\mathbf{x} \circ \mathbf{y}))}{\sqrt{(-\kappa(\mathbf{x} \circ \mathbf{y}))^2 - 1}}(\mathbf{y} + \kappa(\mathbf{x} \circ \mathbf{y})\mathbf{x}). \tag{2.13}$$

The resulting vector $\mathbf{v} = \log_{\mathbf{x}}(\mathbf{y})$ lies in the tangent space $T_{\mathbf{x}}\mathbb{L}_\kappa^n$ (since $\mathbf{v} \circ \mathbf{x} = 0$) and its magnitude is precisely the hyperbolic distance between the two points: $\left\|\log_{\mathbf{x}}(\mathbf{y})\right\|_{\mathcal{L}} = d_{\mathbb{L}}(\mathbf{x}, \mathbf{y})$.

The logarithmic map from the origin simplifies to:

$$\log_{0_{\mathbb{L}_\kappa^n}}(\mathbf{y}) = \frac{\operatorname{arccosh}(\sqrt{\kappa} y_1)}{\sqrt{\kappa y_1^2 - 1}}(0, \overline{\mathbf{y}}). \tag{2.14}$$

Parallel Transport

Parallel transport moves a tangent vector $\mathbf{v} \in T_{\mathbf{x}}\mathbb{L}_\kappa^n$ along the geodesic to another point $\mathbf{y} \in \mathbb{L}_\kappa^n$ while preserving the angle of $\mathbf{v}$ with the geodesic. The resulting vector $P_{\mathbf{x}\to\mathbf{y}}(\mathbf{v}) \in T_{\mathbf{y}}\mathbb{L}_\kappa^n$ is:

$$P_{\mathbf{x}\to\mathbf{y}}(\mathbf{v}) = \mathbf{v} + \frac{\kappa(\mathbf{v} \circ \mathbf{y})}{1 - \kappa(\mathbf{x} \circ \mathbf{y})}(\mathbf{x} + \mathbf{y}). \tag{2.15}$$

Mappings to the Hyperboloid

Beyond the core geodesic operations, it is often useful to map vectors from the ambient Lorentz inner product space $(\mathbb{R}^{n+1}, \circ)$ onto the hyperboloid $\mathbb{L}^n_\kappa$. We define two such operations.

Radial Projection Any positive time-like vector $\mathbf{x} \in \mathbb{R}^{n+1}$ can be projected onto the hyperboloid by scaling it so that its Lorentzian norm becomes $-\frac{1}{\kappa}$. This operation finds the intersection of the span of $\mathbf{x}$ with the hyperboloid.

$$\pi_{\text{rad}}(\mathbf{x}) = \frac{1}{\sqrt{\kappa}\,\|\mathbf{x}\|_{\mathcal{L}}}\text{sign}(x_1)\mathbf{x}. \tag{2.16}$$

Space Orthogonal Projection An alternative way to map any vector $\mathbf{x}$ in $\mathbb{R}^{n+1}$ to the hyperboloid is projecting it along the time axis. This operation preserves the spatial components $\bar{\mathbf{x}}$ of the vector and recomputes the time component x_1 to satisfy the hyperboloid constraint. The resulting vector $\pi_{\text{time}}(\mathbf{x})$ is:

$$\pi_{\text{time}}(\mathbf{x}) = \left(\sqrt{\bar{\mathbf{x}} \cdot \bar{\mathbf{x}} + \frac{1}{\kappa}}, \quad \bar{\mathbf{x}}\right). \tag{2.17}$$

We can visualize these operations in $\mathbb{R}^2$ in 2.3.

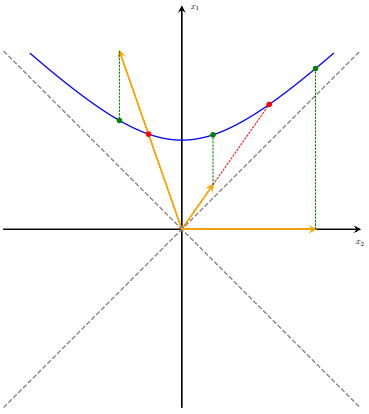


Figure 2.3: The orange arrows (→) are vectors in the Lorentz space, in red (- - -) we can see the radial projection, and in green (- - -) the space orthogonal projection.

2.2.3 The Poincaré Ball Model: $\mathbb{B}^n_\kappa$

The **Poincaré ball model** $\mathbb{B}^n_\kappa$ provides another representation of n-dimensional hyperbolic space with constant negative curvature $-\kappa$ ($\kappa > 0$). It is defined on the open n-dimensional ball of radius $1/\sqrt{\kappa}$ centered at the origin:

$$\mathbb{B}^n_\kappa = \left\{\mathbf{x} \in \mathbb{R}^n \mid \kappa\|\mathbf{x}\|^2 < 1\right\}. \tag{2.18}$$

This model is conformally flat, meaning its Riemannian metric is a scaled version of the Euclidean metric. The scaling is done by the **conformal factor** $\lambda^\kappa_{\mathbf{x}} = \frac{2}{1-\kappa\|\mathbf{x}\|^2}$, which depends on the position $\mathbf{x}$ within the ball. The metric is given by $\mathfrak{g}^{\mathbb{B}}_{\mathbf{x}} = (\lambda^\kappa_{\mathbf{x}})^2\mathfrak{g}^E$, where $\mathfrak{g}^E$ is the standard Euclidean metric. As a point $\mathbf{x}$ approaches the boundary of the ball, its conformal factor $\lambda^\kappa_{\mathbf{x}}$ approaches infinity, causing distances to grow infinitely large.

The algebraic structure of the Poincaré ball is captured by the **gyrovector space** formalism, which provides analogies to vector space operations. The two fundamental operations are **Möbius addition** $\oplus_\kappa$ and **Möbius scalar multiplication** $\otimes_\kappa$.

Hyperbolic Distance

The geodesic distance between two points $\mathbf{x}, \mathbf{y} \in \mathbb{B}_\kappa^n$ is given by the formula:

$$d_{\mathbb{B}}(\mathbf{x}, \mathbf{y}) = \frac{1}{\sqrt{\kappa}} \operatorname{arccosh}\left(1 + 2\frac{\kappa\|\mathbf{x} - \mathbf{y}\|^2}{(1 - \kappa\|\mathbf{x}\|^2)(1 - \kappa\|\mathbf{y}\|^2)}\right). \tag{2.19}$$

Exponential Map

The **exponential map** $\exp_{\mathbf{x}}^\kappa : T_{\mathbf{x}}\mathbb{B}_\kappa^n \to \mathbb{B}_\kappa^n$ maps a tangent vector $\mathbf{v} \in T_{\mathbf{x}}\mathbb{B}_\kappa^n \cong \mathbb{R}^n$ to a point in the ball by moving from $\mathbf{x}$ along a geodesic in the direction of $\mathbf{v}$. It's defined as:

$$\exp_{\mathbf{x}}^\kappa(\mathbf{v}) = \mathbf{x} \oplus_\kappa \left(\tanh\left(\frac{\sqrt{\kappa}}{2}\lambda_{\mathbf{x}}^\kappa\|\mathbf{v}\|\right)\frac{\mathbf{v}}{\|\mathbf{v}\|}\right), \tag{2.20}$$

where the Möbius addition for $\mathbf{x}, \mathbf{y} \in \mathbb{B}_\kappa^n$ is

$$\mathbf{x} \oplus_\kappa \mathbf{y} = \frac{(1 + 2\kappa\langle\mathbf{x}, \mathbf{y}\rangle_E + \kappa\|\mathbf{y}\|^2)\mathbf{x} + (1 - \kappa\|\mathbf{x}\|^2)\mathbf{y}}{1 + 2\kappa\langle\mathbf{x}, \mathbf{y}\rangle_E + \kappa^2\|\mathbf{x}\|^2\|\mathbf{y}\|^2}. \tag{2.21}$$

Logarithmic Map

The **logarithmic map** $\log_{\mathbf{x}}^\kappa : \mathbb{B}_\kappa^n \to T_{\mathbf{x}}\mathbb{B}_\kappa^n$ is the inverse of the exponential map. It finds the tangent vector at $\mathbf{x}$ that points along the geodesic towards $\mathbf{y}$:

$$\log_{\mathbf{x}}^\kappa(\mathbf{y}) = \frac{2}{\sqrt{\kappa}\lambda_{\mathbf{x}}^\kappa} \operatorname{arctanh}\left(\sqrt{\kappa}\|-\mathbf{x} \oplus_\kappa \mathbf{y}\|\right)\frac{-\mathbf{x} \oplus_\kappa \mathbf{y}}{\|-\mathbf{x} \oplus_\kappa \mathbf{y}\|}. \tag{2.22}$$

The magnitude of this vector is equal to the hyperbolic distance between $\mathbf{x}$ and $\mathbf{y}$: $\|\log_{\mathbf{x}}^\kappa(\mathbf{y})\| = d_{\mathbb{B}}(\mathbf{x}, \mathbf{y})$.

Parallel Transport

Parallel transport moves a tangent vector $\mathbf{v} \in T_{\mathbf{x}}\mathbb{B}_\kappa^n$ along the geodesic to $\mathbf{y} \in \mathbb{B}_\kappa^n$. The operation is an isometry between tangent spaces. The transported vector $P_{\mathbf{x}\to\mathbf{y}}^\kappa(\mathbf{v}) \in T_{\mathbf{y}}\mathbb{B}_\kappa^n$ is given by:

$$P_{\mathbf{x}\to\mathbf{y}}^\kappa(\mathbf{v}) = \frac{\lambda_{\mathbf{x}}^\kappa}{\lambda_{\mathbf{y}}^\kappa}\operatorname{gyr}[\mathbf{y}, -\mathbf{x}]\mathbf{v} = \frac{1 - \kappa\|\mathbf{y}\|^2}{1 - \kappa\|\mathbf{x}\|^2}\operatorname{gyr}[\mathbf{y}, -\mathbf{x}]\mathbf{v}. \tag{2.23}$$

The term $\operatorname{gyr}[\mathbf{y}, -\mathbf{x}]$ is the **gyration operator**. $\operatorname{gyr} : \mathbb{B}_\kappa^n \times \mathbb{B}_\kappa^n \to \operatorname{Aut}(\mathbb{B}_\kappa^n, \oplus_\kappa)$. Its expression is:

$$\begin{aligned}\operatorname{gyr}[\mathbf{x}, \mathbf{y}](\mathbf{z}) = \mathbf{z} &- 2\kappa\frac{(\kappa\langle\mathbf{x}, \mathbf{z}\rangle_E\|\mathbf{y}\|_E^2 - \langle\mathbf{y}, \mathbf{z}\rangle_E(1 + 2\kappa\langle\mathbf{x}, \mathbf{y}\rangle_E))\mathbf{x}}{1 + 2\kappa\langle\mathbf{x}, \mathbf{y}\rangle_E + \kappa^2\|\mathbf{x}\|_E^2\|\mathbf{y}\|_E^2} \\ &- 2\kappa\frac{(\kappa\langle\mathbf{y}, \mathbf{z}\rangle_E\|x\|_E^2 + \langle\mathbf{x}, \mathbf{z}\rangle_E)\mathbf{y}}{1 + 2\kappa\langle\mathbf{x}, \mathbf{y}\rangle_E + \kappa^2\|\mathbf{x}\|_E^2\|\mathbf{y}\|_E^2}\end{aligned} \tag{2.24}$$

Related Work 3

The development of hyperbolic neural networks has largely followed a hybrid approach, where data is mapped from the hyperbolic manifold to a Euclidean tangent space to perform operations like linear transformations, and then mapped back using the exponential map Ganea et al., 2018. While this method leverages the well-understood algebra of Euclidean space, the frequent mappings between geometries introduce significant computational overhead due to the evaluation of complex logarithmic and exponential functions. Moreover, these mappings can introduce numerical instability and distortion, potentially limiting the model's ability to fully exploit the representational advantages of hyperbolic geometry Bdeir et al., 2023; Chen et al., 2022

To address these limitations, recent research has explored the design of fully hyperbolic deep learning architectures, where fundamental operations are defined directly on the manifold. This chapter provides an overview of existing approaches to constructing such architectures, focusing specifically on fully connected layers that operate natively in hyperbolic space. We compare two primary strategies that have emerged:

1. **Poincaré-based methods**, which generalize the geometric interpretation of Euclidean affine transformations as signed distances to hyperplanes.
2. **Lorentz-based methods**, which apply standard Euclidean linear operations to the ambient coordinates of the hyperboloid model and then project the result back onto the manifold.

3.1 Poincaré Model: Distance to Hyperplanes

The first approach, pioneered by Ganea et al., 2018 and further refined by Shimizu et al., 2021, constructs a hyperbolic fully connected layer by generalizing the geometric properties of its Euclidean counterpart within the Poincaré ball model.

3.1.1 Interpretation of Euclidean Affine Transformation

An affine transformation, which forms the basis of a standard fully connected layer, can be interpreted geometrically. For an input $\mathbf{x} \in \mathbb{R}^n$, each output dimension y_k of a layer without activation is computed as $y_k = \langle \mathbf{a}_k, \mathbf{x} \rangle_E - b_k$, where $\mathbf{a}_k \in \mathbb{R}^n$ is a weight vector and $b_k \in \mathbb{R}$ is a bias. This equation defines a hyperplane:

$$H_{\mathbf{a}_k, b_k} = \{\mathbf{z} \in \mathbb{R}^n \mid \langle \mathbf{a}_k, \mathbf{z} \rangle_E - b_k = 0\}. \tag{3.1}$$

Notice this hyperplane is at distance $\|\mathbf{a}_k\|_E \, b_k$ from the origin $\mathbf{0}$.

The signed distance from a point $\mathbf{x}$ to this hyperplane is given by $\frac{\langle \mathbf{a}_k, \mathbf{x} \rangle_E - b_k}{\|\mathbf{a}_k\|_E}$. Consequently, the output y_k can be seen as this signed distance, scaled by the norm of the weight vector:

$$y_k = \|\mathbf{a}_k\|_E \cdot \text{sign}(\langle \mathbf{a}_k, \mathbf{x} \rangle_E - b_k) \cdot d_E(\mathbf{x}, H_{\mathbf{a}_k, b_k}). \tag{3.2}$$

Each output dimension y_k corresponds to the signed distance from the input $\mathbf{x}$ to a hyperplane that is orthogonal to the k-th axis of the output space.

3.1.2 Poincaré Fully Connected

This geometric interpretation was extended to the Poincaré ball model, $\mathbb{B}^n_\kappa$, by defining analogues for hyperplanes and the distance to them Ganea et al., 2018. A **Poincaré hyperplane** $\tilde{H}^\kappa_{\mathbf{z},r}$ is defined by an orientation vector $\mathbf{z} \in T_{\mathbf{0}}\mathbb{B}^n_\kappa$ and a scalar distance $r \in \mathbb{R}$ from the origin in a similar way as the Euclidean hyperplane:

$$\tilde{H}^\kappa_{\mathbf{z},r} = \{\mathbf{x} \in \mathbb{B}^n_\kappa \mid \langle \log^\kappa_{\mathbf{q}}(\mathbf{x}), \mathbf{z} \rangle_E = 0\} = \exp^\kappa_{\mathbf{q}}(\{\mathbf{z}\}^\perp). \tag{3.3}$$

Where $\mathbf{q} = \frac{\mathbf{z}}{\|\mathbf{z}\|_E} r$ in $\mathbb{B}^n_\kappa$.

The signed distance from a point $\mathbf{x} \in \mathbb{B}^n_\kappa$ to this hyperplane, denoted $v(\mathbf{x})$, can be computed in closed form Shimizu et al., 2021:

$$v(\mathbf{x}; \mathbf{z}, r) = \frac{2\|\mathbf{z}\|_E}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\lambda^\kappa_{\mathbf{x}} \left\langle \sqrt{\kappa}\mathbf{x}, [\mathbf{z}] \right\rangle_E \cosh(2\sqrt{\kappa}r) - (\lambda^\kappa_{\mathbf{x}} - 1)\sinh(2\sqrt{\kappa}r)\right), \tag{3.4}$$

where $[\mathbf{z}] = \mathbf{z}/\|\mathbf{z}\|_E$.

To transform an input from $\mathbb{B}^n_\kappa$ to an output in $\mathbb{B}^m_\kappa$, this method defines m such hyperplanes in the input space, each parameterized by $(\mathbf{z}_k, r_k)$ for $k = 1, \ldots, m$. This results in m signed distance computations, $\{v_k(\mathbf{x})\}_{k=1}^m$. These scalar values are then used to construct the output vector $\mathbf{y} \in \mathbb{B}^m_\kappa$. A temporary vector $\mathbf{w} \in \mathbb{R}^m$ is formed, where each component is $w_k = \frac{1}{\sqrt{\kappa}} \sinh(\sqrt{\kappa} v_k(\mathbf{x}))$. The final output is then Shimizu et al., 2021:

$$\mathbf{y} = \frac{\mathbf{w}}{1 + \sqrt{1 + \kappa \|\mathbf{w}\|^2_E}}. \tag{3.5}$$

This construction is a principled generalization, as the resulting point $\mathbf{y}$ in the output space lies at a signed hyperbolic distance $v_k(\mathbf{x})$ from the hyperplane orthogonal to the k-th axis. Furthermore, the formulation converges to the Euclidean case (Equation 3.2) as the curvature κ approaches zero Shimizu et al., 2021.

However, this approach has limitations. The repeated evaluation of hyperbolic trigonometric functions for each of the m hyperplanes is computationally intensive, resulting in significantly slower runtimes compared to both Euclidean and other fully hyperbolic methods. Also, the Poincaré model struggles when representing points close to the boundary of the ball in floating point representation.

3.2 Lorentz Model: Euclidean Operations and Projection

Another family of models make use of the Lorentz manifold $\mathbb{L}_\kappa^n$. This methods avoid the distance-to-hyperplane approach. Instead, it applies standard Euclidean linear operations to the ambient coordinates of a hyperbolic vector and then, projects the result back onto the hyperboloid.

This approach originated from the idea of parameterizing Lorentz transformations, which are the isometries of the Lorentz manifold. By relaxing some constraints on the Lorentz transformations, Chen et al., 2022 came up with this new family of methods. Relaxing the constraints on the transformation matrix led to a simpler, though less geometrically principled formulation.

The method operates as follows. An input vector $\mathbf{x} \in \mathbb{L}_\kappa^n$ is treated as a standard euclidean input vector to a layer with parameters W in $\mathbb{R}^{n \times m+1}$ and optionally including bias $\mathbf{b}$ in $\mathbb{R}^{m+1}$, activation function h, dropout or scaling normalization with learnable parameters $\mathbf{v}$ in $\mathbb{R}^{n+1}$,b' in $\mathbb{R}$ and optionally fixed λ in $\mathbb{R}$ yielding one of these expressions:

$$\phi(\mathbf{x}) = \begin{cases} W\mathbf{x} & \text{or} \\ W\text{dropout}(\mathbf{x}) & \text{or} \\ \lambda\sigma(\mathbf{v}^T\mathbf{x} + b')\frac{Wh(\mathbf{x})+\mathbf{b}}{\|Wh(\mathbf{x})+\mathbf{b}\|_E} & \text{or} \\ h(W\mathbf{x} + \mathbf{b}) \end{cases} \tag{3.6}$$

Where σ is the logistic function. The straightforward approach is using a standard linear layer $\phi(\mathbf{x}) = h(W\mathbf{x} + \mathbf{b})$ with output in $\mathbb{R}^m$ as in Bdeir et al., 2023.

The final output $\mathbf{y}$ in $\mathbb{L}_\kappa^m$ is obtained by concatenating a zero to $\phi(\mathbf{x})$ and projecting this new vector back onto the hyperboloid using the space orthogonal projection:

$$\mathbf{y} = \pi_{\text{time}}(\begin{pmatrix} 0 \\ \phi(\mathbf{x}) \end{pmatrix}) = \begin{pmatrix} \sqrt{\|\phi(\mathbf{x})\|_E^2 + \frac{1}{\kappa}} \\ \phi(\mathbf{x}) \end{pmatrix} \tag{3.7}$$

This formulation can be viewed as a standard Euclidean neural network with an additional, non-linear projection step that enforces the hyperbolic geometry constraint. However, this simplicity comes with several drawbacks. The learned transformation is not a Lorentz transformation. The practical implementation by Chen et al., 2022 includes scaling factors and sigmoid functions that result in a learnable function space that does not contain any Lorentz transformations, as the output space is restricted to only some part of the hyperboloid.

This approach effectively treats the hyperbolic input as engineered features for a Euclidean network, applying the same linear map across a space where the metric is non-uniform, ignoring that one unit change in one dimension of the coordinates, causes a non uniform displacement that depends non linearly on the other coordinates.

It also treats the time and spatial components of the input vector the same, which might prevent the full utilization of the properties of hyperbolic geometry.

Notably, Bdeir et al., 2023 also proposed a multinomial logistic regression layer for the Lorentz model using the distance-to-hyperplane interpretation, similar to the Poincaré approach. However, this was not extended to a general-purpose fully connected layer or other layers. It also lacks one key feature of the euclidean linear layer, the parameter that defines the orientation of the hyperplane does not scale the distances of the points to the hyperplane or the displacement of the hyperplane to the origin.

Lorentzian Fully Connected 4

In the last chapter we reviewed how it is defined the fully connected layer in the Poincaré model through generalizing the concept of scaled signed distance to hyperplanes, and in the Lorentz model, through space orthogonal projection.

We now understand the drawbacks of each model. Now, in order to circumvent the disadvantages of each model and to be able to compare the representational capabilities of both manifolds for deep learning, we are going to derive the geometrically grounded definition of the fully connected scaled signed distance to hyperplanes in the Lorentz model.

4.1 Lorentz Distance to hyperplane

The first step is to define what is a hyperplane in the Lorentz model and how can we study it analytically.

4.1.1 Lorentz hyperplanes

Previously, in the Poincaré model, we could define a hyperplane from any orientation vector in $\mathbb{R}^n$ and a bias. we had the advantage that at all points in the manifold, the tangent space was $\mathbb{R}^n$. In the Lorentz model, we will have to change a little our approach.

A **hyperbolic m-plane** of $\mathbb{L}^n_\kappa$ is the intersection of $\mathbb{L}^n_\kappa$ with a (m+1)-dimensional time-like vector subspace of $\mathbb{R}^{n+1}$:

All geodesics in the Lorentz model are 1-planes in the hyperboloid (Ratcliffe, 2019). Therefore, every geodesic is the intersection of a 2 dimensional time-like vector subspace.

The same way that in Euclidean geometry of $\mathbb{R}^n$ a hyperplane is an (n-1)-dimensional plane, in the Lorentz model $\mathbb{L}^n_\kappa$, a **hyperbolic hyperplane** is a hyperbolic (n-1)-plane.

By leveraging the properties of the Lorentzian complement, we can define the Lorentzian hyperplanes in terms of a single vector that will carry information about the orientation of the hyperplane and the displacement from the origin.

Let $\mathbf{w}$ be a space-like vector in $\mathbb{R}^{n+1}$. Then, as its span is a one dimensional space-like vector subspace, the Lorentzian complement of its span is an n-dimensional time-like vector space and its intersection with the hyperboloid is a hyperbolic hyperplane in $\mathbb{L}^n_\kappa$, so, given a space-like vector $\mathbf{w}$ we can define one hyperplane $H_{\mathbf{w}}$ that we call Lorentz orthogonal to $\mathbf{w}$.

$$H_{\mathbf{w}} = \{\mathbf{w}\}^L \cap \mathbb{L}^n_\kappa = \{\mathbf{x} \in \mathbb{L}^n_\kappa \mid \mathbf{x} \circ \mathbf{w} = 0\} \tag{4.1}$$

The orientation of the hyperplane is defined by the direction of the spatial components of $\mathbf{w}$, and the displacement from the origin depends on the angle of inclination of the vector to the space euclidean hyperplane, with zero inclination, the vector is tangent to the origin, and in the limit to 45 degrees, the displacement of the hyperplane tends to the infinite.

Now we are going to derive the formula of the distance from a point $\mathbf{x}$ in the hyperboloid to the hyperplane $H_{\mathbf{w}}$.

4.1.2 Distance to Hyperplane

Let $\mathbf{x} \in \mathbb{L}_\kappa^n$ be a point on the hyperboloid and let $H_{\mathbf{w}}$ be a hyperbolic hyperplane defined by a space-like normal vector $\mathbf{w} \in \mathbb{R}^{n+1}$ as $H_{\mathbf{w}} = \{\mathbf{z} \in \mathbb{L}_\kappa^n \mid \mathbf{z} \circ \mathbf{w} = 0\}$. The distance from $\mathbf{x}$ to $H_{\mathbf{w}}$, denoted $d_{\mathbb{L}}(\mathbf{x}, H_{\mathbf{w}})$, is the length of the shortest path from $\mathbf{x}$ to the hyperplane. This path is a unique geodesic that starts at $\mathbf{x}$ and intersects the hyperplane $H_{\mathbf{w}}$ orthogonally[1].

1: In a Riemannian manifold, the shortest path between a point, in this case $\mathbf{x}$ and a closed submanifold, in this case $H_{\mathbf{w}}$ is always realized by a geodesic that is orthogonal to the submanifold at the point of intersection (Chavel, 2006).

Let $\mathbf{y}$ in $H_{\mathbf{w}}$ be the point of intersection. Then, the tangent vector of the geodesic from $\mathbf{x}$ to $\mathbf{y}$ at $\mathbf{y}$ must be aligned with the normal vector $\mathbf{w}$. Then, the geodesic is contained in a plane that is spanned by $\mathbf{x}$ and $\mathbf{w}$ (Ratcliffe, 2019).

Therefore, as $\mathbf{y}$ is in plane, it can be expressed as a linear combination of $\mathbf{x}$ and $\mathbf{w}$:

$$\mathbf{y} = a\mathbf{x} + b\mathbf{w} \qquad \text{for some } a \text{ and } b \tag{4.2}$$

$$\propto$$

$$\mathbf{y}_{\text{unnormalized}} = \mathbf{x} + \alpha\mathbf{w} \qquad \text{for some } \alpha \tag{4.3}$$

We can substitute by α by multiplying per $\frac{1}{a}$. We can do this if a is non zero, which we can assume because $\mathbf{y}$ will be in the hyperboloid and therefore be time-like, and the span of $\mathbf{w}$ is space-like so it cannot be expressed as

$$\mathbf{y} = c\mathbf{w}$$

Since $\mathbf{y}$ lies on the hyperplane $H_{\mathbf{w}}$, the vector $\mathbf{y}_{\text{unnormalized}}$ must be orthogonal to $\mathbf{w}$:

$$\mathbf{y}_{\text{unnormalized}} \circ \mathbf{w} = 0 \implies (\mathbf{x} + \alpha\mathbf{w}) \circ \mathbf{w} = 0 \tag{4.4}$$

$$\implies (\mathbf{x} \circ \mathbf{w}) + \alpha(\mathbf{w} \circ \mathbf{w}) = 0 \tag{4.5}$$

$$\implies \alpha = -\frac{\mathbf{x} \circ \mathbf{w}}{\|\mathbf{w}\|_{\mathcal{L}}^2}. \tag{4.6}$$

Substituting α back gives:

$$\mathbf{y}_{\text{unnormalized}} = \mathbf{x} - \frac{\mathbf{x} \circ \mathbf{w}}{\|\mathbf{w}\|_{\mathcal{L}}^2}\mathbf{w}. \tag{4.7}$$

To find the point $\mathbf{y} \in \mathbb{L}_\kappa^n$, we must project $\mathbf{y}_{\text{unnormalized}}$ onto the hyperboloid using the radial projection π_{rad} defined in Equation 2.16. First, we

compute the squared Lorentzian norm of $\mathbf{y}_{\text{unnormalized}}$:

$$\begin{aligned}\|\mathbf{y}_{\text{unnormalized}}\|_{\mathcal{L}}^2 &= \left(\mathbf{x} - \frac{\mathbf{x}\circ\mathbf{w}}{\|\mathbf{w}\|_{\mathcal{L}}^2}\mathbf{w}\right)\circ\left(\mathbf{x} - \frac{\mathbf{x}\circ\mathbf{w}}{\|\mathbf{w}\|_{\mathcal{L}}^2}\mathbf{w}\right)\\ &= \|\mathbf{x}\|_{\mathcal{L}}^2 - 2\frac{(\mathbf{x}\circ\mathbf{w})^2}{\|\mathbf{w}\|_{\mathcal{L}}^2} + \frac{(\mathbf{x}\circ\mathbf{w})^2}{(\|\mathbf{w}\|_{\mathcal{L}}^2)^2}\|\mathbf{w}\|_{\mathcal{L}}^2\\ &= \|\mathbf{x}\|_{\mathcal{L}}^2 - \frac{(\mathbf{x}\circ\mathbf{w})^2}{\|\mathbf{w}\|_{\mathcal{L}}^2}.\end{aligned}$$

Since $\mathbf{x}$ is in $\mathbb{L}_\kappa^n$, we have $\|\mathbf{x}\|_{\mathcal{L}}^2 = -1/\kappa$:

$$\|\mathbf{y}_{\text{unnormalized}}\|_{\mathcal{L}}^2 = -\frac{1}{\kappa} - \frac{(\mathbf{x}\circ\mathbf{w})^2}{\|\mathbf{w}\|_{\mathcal{L}}^2} = -\left(\frac{\|\mathbf{w}\|_{\mathcal{L}}^2 + \kappa(\mathbf{x}\circ\mathbf{w})^2}{\kappa\|\mathbf{w}\|_{\mathcal{L}}^2}\right). \quad (4.8)$$

Projecting onto the hyperboloid yields:

$$\mathbf{y} = \pi_{\text{rad}}(\mathbf{y}_{\text{unnormalized}}) = \frac{1}{\sqrt{\kappa}\,\|\mathbf{y}_{\text{unnormalized}}\|_{\mathcal{L}}}\mathbf{y}_{\text{unnormalized}} = \frac{\|\mathbf{w}\|_{\mathcal{L}}}{\sqrt{\|\mathbf{w}\|_{\mathcal{L}}^2 + \kappa(\mathbf{x}\circ\mathbf{w})^2}}\mathbf{y}_{\text{unnormalized}}. \quad (4.9)$$

Now we compute the distance $d_{\mathbb{L}}(\mathbf{x},\mathbf{y}) = \frac{1}{\sqrt{\kappa}}\operatorname{arccosh}(-\kappa(\mathbf{x}\circ\mathbf{y}))$. First we compute $\mathbf{x}\circ\mathbf{y}$:

$$\begin{aligned}\mathbf{x}\circ\mathbf{y} &= \frac{\|\mathbf{w}\|_{\mathcal{L}}}{\sqrt{\|\mathbf{w}\|_{\mathcal{L}}^2 + \kappa(\mathbf{x}\circ\mathbf{w})^2}}\left(\mathbf{x}\circ\left(\mathbf{x} - \frac{\mathbf{x}\circ\mathbf{w}}{\|\mathbf{w}\|_{\mathcal{L}}^2}\mathbf{w}\right)\right) && (4.10)\\ &= \frac{\|\mathbf{w}\|_{\mathcal{L}}}{\sqrt{\|\mathbf{w}\|_{\mathcal{L}}^2 + \kappa(\mathbf{x}\circ\mathbf{w})^2}}\left(\|\mathbf{x}\|_{\mathcal{L}}^2 - \frac{(\mathbf{x}\circ\mathbf{w})^2}{\|\mathbf{w}\|_{\mathcal{L}}^2}\right) && (4.11)\\ &= \frac{\|\mathbf{w}\|_{\mathcal{L}}}{\sqrt{\|\mathbf{w}\|_{\mathcal{L}}^2 + \kappa(\mathbf{x}\circ\mathbf{w})^2}}\left(-\frac{1}{\kappa} - \frac{(\mathbf{x}\circ\mathbf{w})^2}{\|\mathbf{w}\|_{\mathcal{L}}^2}\right) && (4.12)\\ &= -\frac{\sqrt{\|\mathbf{w}\|_{\mathcal{L}}^2 + \kappa(\mathbf{x}\circ\mathbf{w})^2}}{\kappa\|\mathbf{w}\|_{\mathcal{L}}}. && (4.13)\end{aligned}$$

Substituting this into the distance formula:

$$\begin{aligned}d_{\mathbb{L}}(\mathbf{x},H_{\mathbf{w}}) = d_{\mathbb{L}}(\mathbf{x},\mathbf{y}) &= \frac{1}{\sqrt{\kappa}}\operatorname{arccosh}\left(-\kappa\left(-\frac{\sqrt{\|\mathbf{w}\|_{\mathcal{L}}^2 + \kappa(\mathbf{x}\circ\mathbf{w})^2}}{\kappa\|\mathbf{w}\|_{\mathcal{L}}}\right)\right) && (4.14)\\ &= \frac{1}{\sqrt{\kappa}}\operatorname{arccosh}\left(\frac{\sqrt{\|\mathbf{w}\|_{\mathcal{L}}^2 + \kappa(\mathbf{x}\circ\mathbf{w})^2}}{\|\mathbf{w}\|_{\mathcal{L}}}\right) && (4.15)\\ &= \frac{1}{\sqrt{\kappa}}\operatorname{arccosh}\left(\sqrt{1 + \frac{\kappa(\mathbf{x}\circ\mathbf{w})^2}{\|\mathbf{w}\|_{\mathcal{L}}^2}}\right). && (4.16)\end{aligned}$$

Using the identity $\operatorname{arccosh}(\sqrt{1+z^2}) = \operatorname{arcsinh}(|z|)$, we get the distance:

$$d_{\mathbb{L}}(\mathbf{x},H_{\mathbf{w}}) = \frac{1}{\sqrt{\kappa}}\operatorname{arcsinh}\left(\frac{\sqrt{\kappa}|\mathbf{x}\circ\mathbf{w}|}{\|\mathbf{w}\|_{\mathcal{L}}}\right). \quad (4.17)$$

The sign of the term $\mathbf{x} \circ \mathbf{w}$ indicates on which side of the hyperplane the point $\mathbf{x}$ lies[2]. This allows us to define a **signed distance**, that we will use to define the hyperbolic fully connected layer:

$$d_{\mathbb{L}}^{\text{signed}}(\mathbf{x}, H_{\mathbf{w}}) = \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\frac{\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})}{\|\mathbf{w}\|_{\mathscr{L}}}\right). \tag{4.18}$$

The term $\|\mathbf{w}\|_{\mathscr{L}}$ in the denominator of the signed distance formula normalizes the normal vector of the hyperplane. In the euclidean fully connected we saw that the output is not only the signed distance but it is scaled by the magnitude of the hyperplane's normal vector that is indirectly learned as it is a function of the parameters.

In order to modify this distance in a way that is equivalent to that in the Euclidean case we can opt for two options: **(1)** scale the total distance by the Lorentzian norm of the normal vector of the hyperplane $\|\mathbf{w}\|_{\mathscr{L}}$ or, **(2)** unnormalizing the vector orthogonal to the hyperplane $\mathbf{w}$ itself.

It can be proven that both approaches are equivalent in the limit when the curvature tends to 0, more details in A.0.1. We opt to use option **(2)** because it will allow us to simplify our operations heavily when we write our layer in matrix form and when we use Lorentzian activation functions that we will discuss later.

Removing it This gives the model more flexibility, as the network can learn the appropriate magnitude for each weight vector $\mathbf{w}^{(j)}$ during optimization. We will call this:

$$d_{\mathbb{L}}^{\text{signed,scaled}}(\mathbf{x}, H_{\mathbf{w}}) = \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})\right). \tag{4.19}$$

Application to Neural Network Layers

A fully connected layer is mapping an input $\mathbf{x}$ in $\mathbb{L}_{\kappa}^{n}$ to an output $\mathbf{z}$ in $\mathbb{L}_{\kappa}^{m}$. We do this by defining m hyperplanes, each with a normal vector $\mathbf{w}^{(j)}$ in $\mathbb{R}^{n+1}$ for $j = 1, \ldots, m$.

We therefore define the j-th pre-activation, s_j as:

$$s_j = d_{\mathbb{L}}^{\text{signed,scaled}}(\mathbf{x}, H_{\mathbf{w}^{(j)}}) = \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w}^{(j)})\right). \tag{4.20}$$

Notice $\mathbf{x} \circ \mathbf{w}^{(j)} = \mathbf{w}^{(j)T} \operatorname{diag}(-1, 1, ..., 1)\, \mathbf{x}$.
To put it in matrix form lets define the matrix:

$$W = \begin{pmatrix} \text{——}\ \mathbf{w}^{(1)T}\ \text{——} \\ \vdots \\ \text{——}\ \mathbf{w}^{(i)T}\ \text{——} \\ \vdots \\ \text{——}\ \mathbf{w}^{(m)T}\ \text{——} \end{pmatrix} \tag{4.21}$$

2: We can see this by defining a vector $\mathbf{v}$ whose components are related to those of $\mathbf{w}$ as follows:

$$v_1 \equiv -w_1$$
$$v_{2:n} \equiv w_{2:n}$$

Then, the Lorentz inner product can be expressed as a Euclidean dot product:

$$f(\mathbf{x}) = \mathbf{x} \circ \mathbf{w} = \mathbf{x} \cdot \mathbf{v}$$

This function, $f(\mathbf{x})$, is the scaled Euclidean distance to the hyperplane defined by $\mathbf{v}$, with the domain restricted to the hyperboloid. Therefore, as in the Euclidean case, the sign of the term indicates on which side of the hyperplane $\mathbf{x}$ lies.

And let $V = W \text{ diag}(-1, 1, ..., 1)$. Then, we can define $\mathbf{s}$ as:

$$\mathbf{s} = \frac{1}{\sqrt{\kappa}} \text{arcsinh}\left(\sqrt{\kappa} V \mathbf{x}\right) \tag{4.22}$$

Notice $\mathbf{s}$ is in $\mathbb{R}^m$. The pre-activations are not bounded or restricted to the hyperboloid.

To this preactivation we will apply any desired transformations h, including but not limited to activation functions, normalization. This yields the activations $\mathbf{a}$ in $\mathbb{R}^m$:

$$\mathbf{a} = \mathbf{h}(\mathbf{s})$$

To construct the final output point $\mathbf{z} \in \mathbb{L}^m_\kappa$, we take the vector of activations $\mathbf{a}$ and use it to define the spatial components of $\mathbf{z} \in \mathbb{R}^{m+1}$ in such a way that the distance from $\mathbf{z}$ to the hyperplanes orthogonal to each spatial axis in $\mathbb{L}^m_\kappa$ is equal to one distinct entry of $\mathbf{a}$ as in Shimizu et al., 2021. Let $\mathbf{e}^{(j)}$ be the j-th column of the canonical base in $\mathbb{R}^{m+1}$.

Then:

$$\begin{aligned}
a_j &= d_{\mathbb{L}}^{\text{signed,scaled}}(\mathbf{z}, H_{\mathbf{e}^{(j+1)}}) && \text{for } j = 1, ..., m && (4.23)\\
&= \frac{1}{\sqrt{\kappa}} \text{arcsinh}\left(\sqrt{\kappa}(\mathbf{z} \circ \mathbf{e}^{(j+1)})\right) && && (4.24)\\
&= \frac{1}{\sqrt{\kappa}} \text{arcsinh}\left(\sqrt{\kappa} z_{j+1})\right) && && (4.25)\\
\implies z_{j+1} &= \frac{1}{\sqrt{\kappa}} \sinh\left(\sqrt{\kappa} a_j\right) && && (4.26)
\end{aligned}$$

This defined the spatial components of $\mathbf{z}$:

$$\overline{\mathbf{z}} = \frac{1}{\sqrt{\kappa}} \sinh\left(\sqrt{\kappa}\mathbf{a}\right) \tag{4.27}$$

The time component z_1 is constructed to ensure the point lies on the output hyperboloid $\mathbb{L}^m_\kappa$. This is equivalent to applying the space orthogonal projection π_{time} (Equation 2.17) to a vector in $\mathbb{R}^{m+1}$ whose spatial part is $\frac{1}{\sqrt{\kappa}} \sinh\left(\sqrt{\kappa}\mathbf{a}\right)$:

$$z_1 = \sqrt{\frac{1}{\kappa} + \|\overline{\mathbf{z}}\|_E^2} \tag{4.28}$$

In summary, to do a forward pass of a fully connected. Given and input $\mathbf{x}$ in $\mathbb{L}^n$ and V a matrix $m \times n + 1$ whose rows are space-like vectors in $\mathbb{R}^{n+1}$ we:

1. Compute pre-activations:

$$\mathbf{s} = \frac{1}{\sqrt{\kappa}} \text{arcsinh}\left(\sqrt{\kappa} V \mathbf{x}\right)$$

2. Compute activations:

$$\mathbf{a} = \mathbf{h}(\mathbf{s})$$

3. Compute space components of the output:

$$\overline{\mathbf{z}} = \frac{1}{\sqrt{\kappa}} \sinh\left(\sqrt{\kappa}\mathbf{a}\right)$$

4. Compute the time component of the output:

$$z_1 = \sqrt{\frac{1}{\kappa} + \|\overline{\mathbf{z}}\|_E^2}$$

Case: $\mathbf{h} = \mathbf{ReLU}$

Let the activation function $\mathbf{h}$ be the element-wise Rectified Linear Unit $\mathbf{h}(\mathbf{s}) = \text{ReLU}(\mathbf{s}) = \max(\mathbf{0}, \mathbf{s})$. The full transformation from the input $\mathbf{x}$ to the spatial components of the output $\overline{\mathbf{z}}$ is:

$$\mathbf{s} = \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\sqrt{\kappa} V \mathbf{x}\right) \tag{4.29}$$

$$\mathbf{a} = \text{ReLU}(\mathbf{s}) \tag{4.30}$$

$$\overline{\mathbf{z}} = \frac{1}{\sqrt{\kappa}} \sinh\left(\sqrt{\kappa}\mathbf{a}\right) \tag{4.31}$$

This chain of operations simplifies significantly due ReLU commuting with a multiplication by a positive scalar $\sqrt{k}$ and with arcsinh[3]:

$$\mathbf{a} = \text{ReLU}(\mathbf{s}) = \text{ReLU}\left(\frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}(\sqrt{\kappa} V \mathbf{x})\right) \tag{4.32}$$

$$= \frac{1}{\sqrt{\kappa}} \text{ReLU}(\operatorname{arcsinh}(\sqrt{\kappa} V \mathbf{x})) \tag{4.33}$$

$$= \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}(\text{ReLU}(\sqrt{\kappa} V \mathbf{x})) \tag{4.34}$$

When we compute $\overline{\mathbf{z}}$:

$$\overline{\mathbf{z}} = \frac{1}{\sqrt{\kappa}} \sinh\left(\sqrt{\kappa}\mathbf{a}\right) \tag{4.35}$$

$$= \frac{1}{\sqrt{\kappa}} \sinh\left(\sqrt{\kappa} \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}(\text{ReLU}(\sqrt{\kappa} V \mathbf{x}))\right) \tag{4.36}$$

$$= \text{ReLU}(V\mathbf{x}) \tag{4.37}$$

This is a remarkable simplification. The expensive arcsinh and sinh operations completely cancel out, leaving a simple matrix-vector product followed by a ReLU activation. This avoids the numerical and computational overhead of the hyperbolic trigonometric functions, making the layer significantly more efficient as we will see in the experiments.

Lorentzian activation functions

The previous simplification was possible due to properties specific to the ReLU function. Activations such as the Leaky ReLU, GELU, Swish, and others do not allow this.

However, in deep learning, the choice of activation function, is not strictly grounded in deep theoretical principles. Given the conditions that it

3: Case 1:

$$x \geqslant 0 \Rightarrow \operatorname{arcsinh}(x) \geqslant 0$$

$$\Rightarrow \text{ReLU}(\operatorname{arcsinh}(x)) = \operatorname{arcsinh}(x) = \operatorname{arcsinh}(\text{ReLU}(x)).$$

Case 2:

$$x < 0 \Rightarrow \operatorname{arcsinh}(x) < 0$$

$$\Rightarrow \text{ReLU}(\operatorname{arcsinh}(x)) = 0 = \operatorname{arcsinh}(\text{ReLU}(x))$$

Therefore

$$\Rightarrow \text{ReLU}(\operatorname{arcsinh}(x)) = \operatorname{arcsinh}(\text{ReLU}(x)).$$

introduces a non linearity[4] in the operation, and, to be differentiable almost everywhere, there are no other necessary conditions[5].

The choice is is primarily empirically driven. Many of these functions have demonstrated good capabilities facing vanishing gradients, and future activation functions might be more robust against this problem. To utilize these properties and in order to improve computational efficiency and numerical stability, we define a family of activation functions such that, when applied to the pre-activations, they simplify the operations in the same way as in the ReLU case such that the final expression for $\overline{\mathbf{z}}$ is $f(V\mathbf{x})$ for any activation function f.

Let $f : \mathbb{R} \to \mathbb{R}$ be a real valued function and κ a positive real number. Then, we define the Lorentzian version of f:

$$f_{\text{Lorentzian},\kappa}(x) = \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}(\sqrt{\kappa} f(\frac{1}{\sqrt{\kappa}} \sinh(\sqrt{\kappa} x))) \tag{4.38}$$

Then, given the elementwise Lorentzian version of an activation function as activation we have:

$$\begin{aligned}
\overline{\mathbf{z}} &= \frac{1}{\sqrt{\kappa}} \sinh(\sqrt{\kappa} \mathbf{f}_{\text{Lorentzian},\kappa}(\frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}(\sqrt{\kappa} V\mathbf{x}))) && (4.39)\\
&= \frac{1}{\sqrt{\kappa}} \sinh(\sqrt{\kappa} \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}(\sqrt{\kappa} \mathbf{f}(\frac{1}{\sqrt{\kappa}} \sinh(\sqrt{\kappa} x)(\frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}(\sqrt{\kappa} V\mathbf{x}))))) && (4.40)\\
&= f(V\mathbf{x}) && (4.41)
\end{aligned}$$

Note that the Lorentz version of ReLU is the same as the original ReLU.

How do the Lorentzian activations compare to the original activation functions? It can be proved that when the curvature κ tends to zero, the Lorentzian function tends to the original function.

In the literature, common values for curvature range between 0 and 1.

As we can see in Figure 4.1, for most of ReLU like functions its Lorentzian versions and them behave very similar.

4: Some earlier works argued that in hyperbolic deep learning, activation functions were not necessary because the transformations in hyperbolic space were already non linear

5: Another beneficial condition is that the function is not saturated and allows the gradient to flow better in the backwards pass.

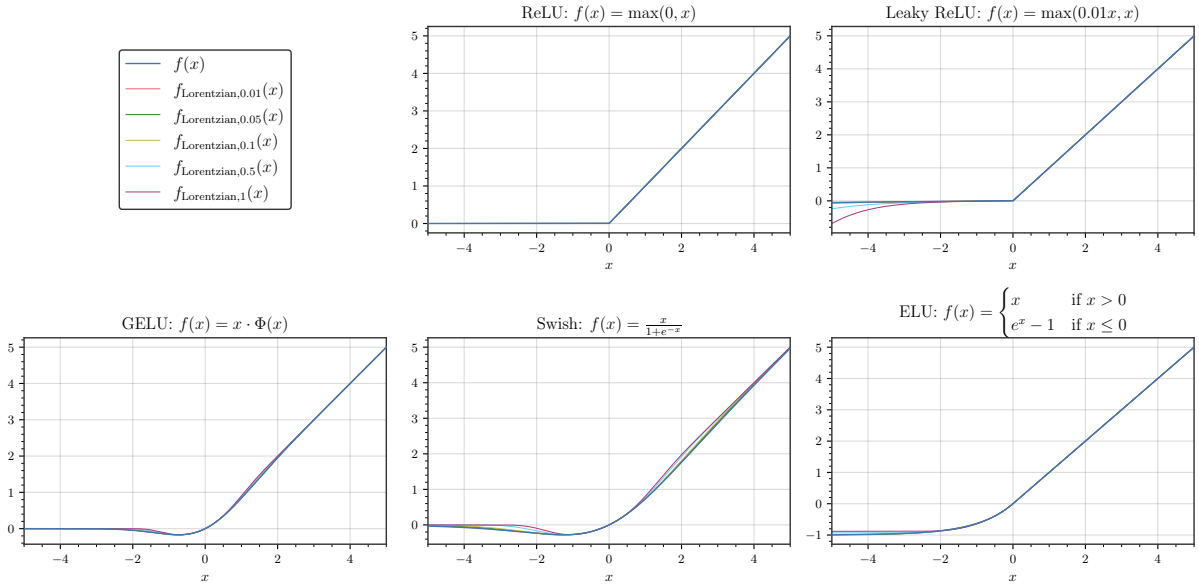


Figure 4.1: A figure of some common activation functions and their Lorentzian version. We can see that they are very similar, and as the curvature decreases, $f_{\text{Lorentzian},\kappa}$ approximates more to f

4.2 Parametrization of tangent vector w

In the previous section, we defined the Lorentzian fully connected layer using weight vectors $\mathbf{w} \in \mathbb{R}^{n+1}$ that define the hyperplanes. However, learning these vectors directly is challenging for two main reasons: (1) the space of valid (space-like) vectors is a non-convex cone, and (2) the components of $\mathbf{w}$ have a complex geometric relationship with the hyperplane's orientation and displacement. To address this, we reparametrize each space-like vector $\mathbf{w}$ using parameters with clear geometric and Euclidean interpretations.

For each hyperplane $H_{\mathbf{w}}$, we introduce a **direction vector** $\mathbf{u} \in \mathbb{R}^n$ and a **bias** parameter $b \in \mathbb{R}$. The vector $\mathbf{u}$ encodes both the orientation of the hyperplane's normal vector and also the distance scaling factor, while b controls its displacement from the origin. We construct the components of $\mathbf{w} = (w_1, \overline{\mathbf{w}})$ in $\mathbb{R}^{n+1}$ from $\mathbf{u}$ and b as follows:

The signed distance from the hyperplane $H_{\mathbf{w}}$ to the origin $\mathbf{0}_{\mathbb{L}^n_\kappa}$ must be equal to $b \|\mathbf{u}\|_E$:

$$b \|\mathbf{u}\|_E = d^{\text{signed}}_{\mathbb{L}}(\mathbf{0}_{\mathbb{L}^n_\kappa}, H_{\mathbf{w}}) \tag{4.42}$$

The orientation of $H_{\mathbf{w}}$ is defined by the spatial components $\overline{w}$ of $\mathbf{w}$. So, we will set $\overline{w}$ to be proportional to $\mathbf{u}$:

$$\mathbf{u} = \alpha \overline{\mathbf{w}} \quad \text{for some } \alpha \text{ in } \mathbb{R} \tag{4.43}$$

The scaling factor for the distance is intended to be the magnitude of $\mathbf{u}$. In our formulation of the scaled distance, the scaling is done to $\mathbf{w}$[6] multiplying it by its Lorentzian norm. Therefore, we enforce the constraint

6: Remember we scaled $\mathbf{w}$ instead of the distance itself. We discussed this before Equation 4.19)

that the Lorentzian norm of $\mathbf{w}$ equals the Euclidean norm of $\mathbf{u}$:

$$\|\mathbf{w}\|_{\mathcal{L}} = \|\mathbf{u}\|_E \tag{4.44}$$

To find $\mathbf{w}$ such that relates to $\mathbf{u}$ and b through these conditions, we will follow through this steps.

First we will find a point in $\mathbb{L}^n_\kappa$ that is at distance $b\,\|\mathbf{u}\|_E$ from $\mathbf{0}_{\mathbb{L}^n_\kappa}$ in the spatial direction of $\mathbf{u}$ using the exponential map from 0 (Equation 2.12) of a tangent vector with direction $\mathbf{u}$in the spatial dimensions:

$$\mathbf{z} = \exp_{\mathbf{0}_{\mathbb{L}^n_\kappa}}((0, b\mathbf{u})) = \left(\frac{1}{\sqrt{\kappa}}\cosh(\sqrt{\kappa}b\,\|\mathbf{u}\|_E), \quad \sinh(\sqrt{\kappa}b\,\|\mathbf{u}\|_E)\frac{\mathbf{u}}{\sqrt{\kappa}\,\|\mathbf{u}\|_E}\right) \tag{4.45}$$

This vector $\mathbf{z}$ is a point in the hyperboloid with spatial components proportional to $\mathbf{u}$. To find $\mathbf{w}$ we will find a vector tangent to $\mathbf{z}$ whose spatial components are still proportional to $\mathbf{u}$.

We can achieve that by reflecting across the light cone [7]. There are two possible reflections:

$$\rho_{\text{light }+}(\mathbf{x}) = (\|\bar{\mathbf{x}}\|_E\,, \quad x_1\frac{\bar{\mathbf{x}}}{\|\bar{\mathbf{x}}\|_E}) \tag{4.46}$$

$$\rho_{\text{light }-}(\mathbf{x}) = (-\|\bar{\mathbf{x}}\|_E\,, \quad -x_1\frac{\bar{\mathbf{x}}}{\|\bar{\mathbf{x}}\|_E}) \tag{4.47}$$

They the same vector up to a proportionality constant of -1. One will give us positive space-like vectors and the other negative space-like vectors. This vector will be proportional to $\mathbf{w}$ that defines our hyperplane.

The sign indicates to which direction will the signed distances be positive and negative. Positive $\mathbf{w}$ will yield negative values towards the origin and vice versa. Therefore we will assign which one we use depending on the sign of b:

$$\rho(\mathbf{x}, b) = \begin{cases} \rho_{\text{light }+}(\mathbf{x}) & \text{if } b \geqslant 0 \\ \rho_{\text{light }-}(\mathbf{x}) & \text{if } b < 0 \end{cases} \quad = \rho_{\text{light }+}(\mathbf{x})\,\text{sign}(b) \tag{4.48}$$

We apply the reflection to $\mathbf{z}$ and b:

$$\mathbf{w}_{\text{unscaled}} = \rho(\mathbf{z}, b) \tag{4.49}$$

$$= \left(\frac{1}{\sqrt{\kappa}}\sinh(\sqrt{\kappa}b\,\|\mathbf{u}\|_E), \quad \frac{1}{\sqrt{\kappa}}\cosh(\sqrt{\kappa}b\,\|\mathbf{u}\|_E)\frac{\mathbf{u}}{\|\mathbf{u}\|_E}\right) \tag{4.50}$$

This vector $\mathbf{w}_{\text{unscaled}}$ is by construction orthogonal to $\mathbf{z}$, and thus defines a hyperplane $H_{\mathbf{w}_{\text{unscaled}}}$ that contains the point $\mathbf{z}$. The spatial components of $\mathbf{w}_{\text{unscaled}}$ are proportional to $\mathbf{u}$, which sets the orientation of the hyperplane.

Notice[8] that $\mathbf{w}_{\text{unscaled}} = \|\rho(\mathbf{x}, b)\|_{\mathcal{L}} = -\|\mathbf{x}\|_{\mathcal{L}} = \frac{1}{\sqrt{k}}$:

7: See that $\mathbf{x} \perp_{\mathcal{L}} \rho(\mathbf{x}, b)$:

$$\begin{aligned}\mathbf{x} \circ \rho_{\text{light }+}(\mathbf{x}) = \mathbf{x} \circ \rho_{\text{light }-}(\mathbf{x}) &= -x_1\|\bar{\mathbf{x}}\|_E + x_1\frac{\bar{\mathbf{x}}\cdot\bar{\mathbf{x}}}{\|\bar{\mathbf{x}}\|_E} \\ &= -x_1\|\bar{\mathbf{x}}\|_E + x_1\|\bar{\mathbf{x}}\|_E = 0\end{aligned}$$

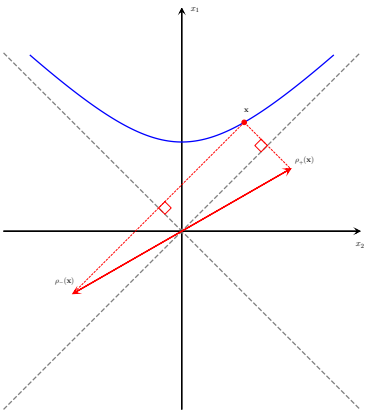


Figure 4.2: Reflections through the light-cone in 2D Lorentzian space. Notice the projection lines are euclidean orthogonal to the light cone.

8: It follows from:

$$\begin{aligned}\|\rho(\mathbf{x}, b)\|^2_{\mathcal{L}} &= \|\rho_{\text{light}+}(\mathbf{x})\,\text{sign}(b)\|^2_{\mathcal{L}} \\ &= -\|\bar{\mathbf{x}}\|^2_E + x_1^2\frac{\bar{\mathbf{x}}\cdot\bar{\mathbf{x}}}{\|\bar{\mathbf{x}}\|^2_E} \\ &= -\bar{\mathbf{x}}\cdot\bar{\mathbf{x}} + x_1^2 = -\|\mathbf{x}\|^2_{\mathcal{L}}\end{aligned}$$

To satisfy our constraint from Equation 4.44 that the final vector $\mathbf{w}$ has a Lorentzian norm equal to the Euclidean norm of $\mathbf{u}$, we must scale $\mathbf{w}_{\text{unscaled}}$. First, we multiply by $\sqrt{\kappa}$ to obtain a vector with Lorentzian norm of 1. Then, we multiply by $\|\mathbf{u}\|_E$ to achieve the desired final norm.

$$\mathbf{w} = \sqrt{\kappa}\,\|\mathbf{u}\|_E\,\mathbf{w}_{\text{unscaled}} \tag{4.51}$$

Substituting the expression for $\mathbf{w}_{\text{unscaled}}$ gives the components of our final parametrized vector $\mathbf{w} = (w_1, \overline{\mathbf{w}})$:

$$w_1 = \sqrt{\kappa}\,\|\mathbf{u}\|_E \left(\frac{\sinh(\sqrt{\kappa} b\,\|\mathbf{u}\|_E)}{\sqrt{\kappa}}\right) = \|\mathbf{u}\|_E \sinh(\sqrt{\kappa} b\,\|\mathbf{u}\|_E) \tag{4.52}$$

$$\overline{\mathbf{w}} = \sqrt{\kappa}\,\|\mathbf{u}\|_E \left(\frac{\cosh(\sqrt{\kappa} b\,\|\mathbf{u}\|_E)}{\sqrt{\kappa}} \frac{\mathbf{u}}{\|\mathbf{u}\|_E}\right) = \cosh(\sqrt{\kappa} b\,\|\mathbf{u}\|_E)\mathbf{u} \tag{4.53}$$

This gives us a final expression for $\mathbf{w}$ in terms of our desired parameters $\mathbf{u}$ in $\mathbb{R}^n$ and b in $\mathbb{R}$.

Finally, we verify that this parametrization satisfies the three initial conditions.

1. **Distance to origin:** The signed distance from the origin $\mathbf{0}_{\mathbb{L}_\kappa^n}$ to the hyperplane $H_{\mathbf{w}}$ is:

$$\begin{aligned} d_{\mathbb{L}}^{\text{signed}}(\mathbf{0}_{\mathbb{L}_\kappa^n}, H_{\mathbf{w}}) &= \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\frac{\sqrt{\kappa}(\mathbf{0}_{\mathbb{L}_\kappa^n} \circ \mathbf{w})}{\|\mathbf{w}\|_{\mathcal{L}}}\right) \\ &= \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\frac{\sqrt{\kappa}(-w_1/\sqrt{\kappa})}{\|\mathbf{u}\|_E}\right) = \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(-\frac{w_1}{\|\mathbf{u}\|_E}\right) \\ &= \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(-\frac{\|\mathbf{u}\|_E \sinh(\sqrt{\kappa} b\,\|\mathbf{u}\|_E)}{\|\mathbf{u}\|_E}\right) \\ &= \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(-\sinh(\sqrt{\kappa} b\,\|\mathbf{u}\|_E)\right) \\ &= -\frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\sinh(\sqrt{\kappa} b\,\|\mathbf{u}\|_E)\right) = -b\,\|\mathbf{u}\|_E \end{aligned}$$

2. **Proportionality:** The spatial part $\overline{\mathbf{w}}$ is $\overline{\mathbf{w}} = \cosh(\sqrt{\kappa} b\,\|\mathbf{u}\|_E)\mathbf{u}$. This is directly proportional to $\mathbf{u}$. This condition is met.
3. **Lorentzian norm:** The norm was set by construction.

$$\begin{aligned} \|\mathbf{w}\|_{\mathcal{L}} = \left\|\sqrt{\kappa}\,\|\mathbf{u}\|_E\,\mathbf{w}_{\text{unscaled}}\right\|_{\mathcal{L}} &= \sqrt{\kappa}\,\|\mathbf{u}\|_E\,\|\mathbf{w}_{\text{unscaled}}\|_{\mathcal{L}} \\ &= \sqrt{\kappa}\,\|\mathbf{u}\|_E \left(\frac{1}{\sqrt{\kappa}}\right) = \|\mathbf{u}\|_E \end{aligned}$$

   This condition is met.

4.3 Lorentz fully connected

We are now ready to define the Lorentz fully connected layer. This layer combines the reparametrization of the hyperplane's normal vector $\mathbf{w}$ in terms of a direction vector $\mathbf{u}$ and a bias b with the scaled signed distance formula.

4.3.1 Definition

A Lorentz fully connected layer maps an input from one hyperbolic space to another given some parameters.

- **Input:** A point $\mathbf{x} \in \mathbb{L}_\kappa^n$.
- **Parameters:** A weight matrix $U \in \mathbb{R}^{m \times n}$ and a bias vector $\mathbf{b} \in \mathbb{R}^m$. Each row $\mathbf{u}^{(j)}$ of U and scalar b_j from $\mathbf{b}$ parametrize one of the m hyperplanes that define the mapping.
- **Output:** A point $\mathbf{z} \in \mathbb{L}_\kappa^m$.

The forward pass is performed in the following steps:

1. **Compute matrix V.** For each output feature $j \in \{1, \dots, m\}$, the corresponding j-th row of the matrix $V \in \mathbb{R}^{m \times (n+1)}$, denoted $\mathbf{v}^{(j)}$, is constructed from the j-th row of U, denoted $\mathbf{u}^{(j)}$, and the j-th element of $\mathbf{b}$, denoted b_j:

$$\mathbf{v}^{(j)} = \left(-\left\|\mathbf{u}^{(j)}\right\|_E \sinh(\sqrt{\kappa} b_j \left\|\mathbf{u}^{(j)}\right\|_E), \quad \cosh(\sqrt{\kappa} b_j \left\|\mathbf{u}^{(j)}\right\|_E) \mathbf{u}^{(j)T} \right)$$

2. **Compute the spatial components of the output $\overline{\mathbf{z}}$.** The spatial components of the output, $\overline{\mathbf{z}} \in \mathbb{R}^m$, are computed by applying the matrix V to the input $\mathbf{x}$, followed by an element-wise activation function $\mathbf{h}$.

$$\overline{\mathbf{z}} = \mathbf{h}(V\mathbf{x})$$

3. **Compute the time component z_1 and form the final output.**

$$z_1 = \sqrt{\frac{1}{\kappa} + \|\overline{\mathbf{z}}\|_E^2}$$

The final output is constructed by combining the time and spatial components:

$$\mathbf{z} = \begin{pmatrix} z_1 \\ \overline{\mathbf{z}} \end{pmatrix} \in \mathbb{L}_\kappa^m$$

4.3.2 Accelerating Inference

During inference, the parameters of the neural network are fixed. In our proposed Lorentzian fully connected layer, the first step of the forward pass involves computing the matrix V from the learned parameters U and $\mathbf{b}$. This computation, described in Section 4.3, does not depend on the input $\mathbf{x}$. Consequently, we can significantly accelerate inference by pre-computing and caching the matrix V, thereby avoiding the repeated evaluation of hyperbolic sine and cosine functions for every forward pass.

A naive caching approach would require storing both the original parameters $(U, \mathbf{b})$ and the cached matrix V. This would effectively double the memory required for the model's parameters, which may be undesirable. However, the mapping from $(\mathbf{u}, b)$ to the corresponding row of V is invertible. This means that once the model is trained, we can discard the original parameters U and $\mathbf{b}$ and store only the matrix V. This strategy provides the full speed-up of caching without any additional memory overhead.

Furthermore, this approach remains flexible for future training or fine-tuning. If, for instance, a distribution shift occurs and we wish to perform continual learning, we can recover the original parameters U and $\mathbf{b}$ directly from the stored matrix V and resume the training process. Let $\mathbf{v}$ be a row of the matrix V, and let $\mathbf{w}$ be the corresponding space-like normal vector obtained by flipping the sign of the time component of $\mathbf{v}$. That is, if $\mathbf{v} = (v_1, \overline{\mathbf{v}})$, then $\mathbf{w} = (-v_1, \overline{\mathbf{v}}) = (w_1, \overline{\mathbf{w}})$. The original parameters $\mathbf{u}$ and b can be recovered from $\mathbf{w}$ using the following equations[9]:

9: Details of the derivation in the Proposition A.0.2

$$\mathbf{u} = \frac{\overline{\mathbf{w}}}{\|\overline{\mathbf{w}}\|_E} \|\mathbf{w}\|_{\mathcal{L}} = \frac{\overline{\mathbf{v}}}{\|\overline{\mathbf{v}}\|_E} \|\mathbf{v}\|_{\mathcal{L}} \tag{4.54}$$

$$b = -\frac{1}{\sqrt{\kappa}\,\|\mathbf{w}\|_{\mathcal{L}}} \operatorname{arcsinh}\left(\frac{w_1}{\|\mathbf{w}\|_{\mathcal{L}}}\right) = \frac{1}{\sqrt{\kappa}\,\|\mathbf{v}\|_{\mathcal{L}}} \operatorname{arcsinh}\left(\frac{v_1}{\|\mathbf{v}\|_{\mathcal{L}}}\right) \tag{4.55}$$

An alternative method to accelerate inference would be to learn the matrix V directly, bypassing the parametrization with U and $\mathbf{b}$ altogether. In this paradigm, each row of V would be treated as a parameter vector constrained to lie on the Riemannian manifold $\mathcal{C}$, which is the set of all valid space-like normal vectors. The Riemannian metric on $\mathcal{C}$ would be the one inherited from the parametrization, where $(\mathbf{u}, b)$ are considered coordinates in $\mathbb{R}^{n+1}$ with the standard Euclidean metric.

This approach falls under the domain of **learning on Riemannian manifolds** that we viewed in Section 2.1.4. To implement an optimization algorithm like Riemannian Stochastic Gradient Descent, one would require the expression for the exponential map on the manifold $\mathcal{C}$. The exponential map is necessary to project the gradient of the parameters that is in the tangent space at some point, back to the manifold. However, finding a closed-form expression for the exponential map of this particular manifold might not be straightforward.

4.3.3 Relation to prior work.

Our forward process is analogous to prior work in the Poincaré model but overcomes some limitations in efficiency and generality. For instance, Shimizu et al., 2021 omitted activation functions entirely, arguing that hyperbolic operations are inherently non-linear. While van Spengler et al., 2023 later demonstrated that activations improve performance, their approach differed from us. They applied ReLU by mapping points to the tangent space at the origin and back using the logarithmic and exponential maps at $\mathbf{0}_{\mathbb{L}^n_\kappa}$. This procedure is not only computationally expensive but is also only equivalent to our method for the specific case of ReLU (and Leaky ReLU).

To create a new point that follows that the distance to the axis hyperplanes is equal to the values of some other score vector, Poincaré methods require to compute said point in the Lorentz model first as described in Equations 4.27 and 4.28, and then to project it to Poincaré. This is a computational cost that could be avoided by computing natively in the Lorentz model. Their reliance on log/exp maps for activations introduces a significant computational bottleneck. They interpret activations as acting on tangent space coordinates, whereas our approach applies them directly to the hyperbolic distances to axis hyperplanes.

After the simplifications made by using Lorentz activations, our output in Equation 4.41 looks very similar to the one proposed by Chen et al., 2022 in Equation 3.7 with $\phi(\mathbf{x}) = h(V\mathbf{x})$. There are however some important differences. First, our derivation shows a geometrically principled way to derive this formula, making clear which should be the choice of ϕ among all formulas proposed in Chen et al., 2022 and in following work such as Bdeir et al., 2023 and Ayubcha et al., 2024 that never explicitly use a linear function without bias before an activation. Second, and more importantly, our weight matrix V is constrained to be composed of **space-like vectors**. In contrast, Chen et al., 2022 allows any matrix in $\mathbb{R}^{m\times n}$ and optimizes it with standard gradient descent. This assumes a Euclidean metric and ignores the distinct geometric role of the time dimension, which our formulation correctly handles.

Bdeir et al., 2023 develop a multinomial logistic regression in the Loretnz model. However they to not extend their work to define a fully connected layer and use the one from Chen et al., 2022 instead. It is nevertheless, different, the final expression used in their multinomial logistic regression than in our Fully connected, as they use the signed hyperbolic distance from the point in the Lorentz manifold to the hyperplane, but they do not scale it. This stops the model from learning a different magnitude per class/output dimension.

The parametrization that we derived, shares many similarities with some others seen in the literature. Mishne et al., 2023 reparametrize hyperplanes in the Lorentzian model with a vector in the tangent space $\overline{\mathbf{z}}$ at $\mathbf{0}_{\mathbb{L}^n_\kappa}$ that defines the orientation of the hyperplane and with a displacement parameter a, However, The total displacement of the hyperplane from the origin $\mathbf{0}_{\mathbb{L}^n_\kappa}$ should not be a, but $a\,\|\overline{\mathbf{z}}\|_E$. They did not extended their parametrization to Lorentz manifolds of $\kappa! = 1$.

Another closely-related parametrization appears in Bdeir et al., 2023. They extend the work to Lorentz manifolds of different curvature, but they still have the problem that the displacement of the hyperplane is not scaled by the magnitude of the direction vector.

4.3.4 Extending the Fully Connected: Convolutions, Transformers, and More

While linear layers form the fundamental building blocks of deep learning, our geometrically principled Lorentzian fully connected layer serves as a powerful drop-in replacement for its Euclidean counterparts. This interchangeability allows us to readily extend our work and construct more complex, state-of-the-art architectures such as Convolutional Neural Networks (CNNs), Transformers, and Graph Neural Networks (GNNs) in hyperbolic space. By substituting standard linear transformations with our layer, we can leverage the vast body of architectural research while harnessing the representational advantages of Lorentzian geometry.

For instance, we can build a **Lorentzian CNN** to process grid-like data, such as images. The core idea is to replicate the convolution operation within the hyperboloid. This involves partitioning the input into local patches, where each patch is a collection of points in $\mathbb{L}^n_\kappa$. These points

are then concatenated into a single vector representation in a higher-dimensional hyperbolic space. Given methods for vector concatenation and splitting that properly preserve the Lorentzian norm (Shimizu et al., 2021), we can apply our Lorentzian fully connected layer to these patch representations, effectively acting as a shared hyperbolic filter. This allows us to create deep convolutional networks that operate entirely within the Lorentz model, inheriting the efficiency and geometric grounding of our layer.

The same principle of substitution applies to other prominent architectures. In the realm of sequence modeling, we can construct **hyperbolic Transformers** by integrating our layer into the feed-forward sub-modules. This complements existing work on hyperbolic attention mechanisms, enabling the creation of fully Lorentzian Transformer blocks (Gulcehre et al., 2018). Similarly, for graph-structured data, our layer can replace the linear transformations in **Graph Convolutional Networks** or **Graph Attention Networks**. This facilitates learning node embeddings in hyperbolic space—a particularly powerful approach for data with inherent hierarchical or tree-like structures.

In essence, although our study has focused on the fundamental properties of a single layer, its implications are far-reaching. The Lorentzian fully connected layer we have developed is not an isolated construct but a gateway to "hyperbolizing" a wide spectrum of deep learning models. By providing a direct and efficient replacement for standard FC layers, our work enables researchers to immediately leverage decades of architectural innovation, combining it with the growing body of research in hyperbolic neural networks to build more powerful and geometrically-aware models.

Summary

We introduce the Lorentzian fully connected layer, defined via signed distances to hyperplanes in the Lorentz model. Each layer is parameterized by a set of space-like vectors, constructed from Euclidean parameters with clear geometric meaning. We derive a closed-form expression for the layer's output and show that, with Lorentz activations, the computation simplifies significantly.

5 Experiments

In this chapter, we explain the conducted experiments and discuss their results. The first experiment is a benchmark of the runtimes of different alternatives for the Fully connected layer on synthetic data. The second is a benchmark to compare different MLR definitions as classifier heads on fine-grained image classification tasks.

5.1 Experiment 1: Benchmarking Layer Runtimes

This experiment systematically benchmarks the GPU runtime performance of various implementations of fully connected and multinomial logistic regression layers. We want to compare the computational efficiency of the Lorentz layer we have defined with the Euclidean Linear layer and with the alternatives discussed in the Related Work Chapter 3.

5.1.1 Setup

The performance is measured for both forward and backward passes using synthetic data. Each time, we sample a batch of random vectors from a normal distribution and vary key parameters that influence computation: input dimension, output dimension, and batch size. The input dimension is defined as the dimension of the manifold where the input data lives (the original dimension of the sampled vectors) [1]. Then we compute the corresponding exponential map from 0 of the input sampled vector[2], and finally we pass this vector to the benchmarked layer. We only measure this last step, not including the exponential map.

1: Consequently, for an input tensor in an n-dimensional manifold, the tensor will have n coordinates in the Euclidean and Poincaré models, but $n + 1$ coordinates in the Lorentz model. The definition is similar for the output dimension, therefore, Lorentz models will have one more coordinate in their inputs and outputs.

2: For Euclidean manifold, the identity

To have more reliable estimations of the runtime, we average over 50 iterations, then we get rid of outliers outside 1.5 of the interquartile range. We perform all the computations in a single run with the same GPU, Nvidia A100 and do 10 iterations of warmup before doing the 50 benchmarked iterations. We report the average time and the standard deviation. As the standard deviation is so small, the error bars cannot be seen in the plot.

We distinguish two types of methods, the ones that can be used as Classification heads, and we will call Multinomial Logistic Regressors (MLRs), and the ones that map between manifolds of different dimensions we call Fully connected (FC). The layers that we are evaluating are:

Euclidean Linear: PyTorch `nn.Linear` layer, serving as the primary performance baseline. We evaluate with bias and without bias parameters. When comparing with MLRs, we benchmark without an activation function. When we compare with FC, we benchmark with ReLU.

Lorentz FC: The implementation of our Fully connected layer. We benchmark both the standard version and cached version following Section 4.3.2. When comparing to MLRs, we benchmark not the full layer but only the computation of the scaled signed distance to the hyperplanes based on the parameters. When comparing to FC, we benchmark the full layer.

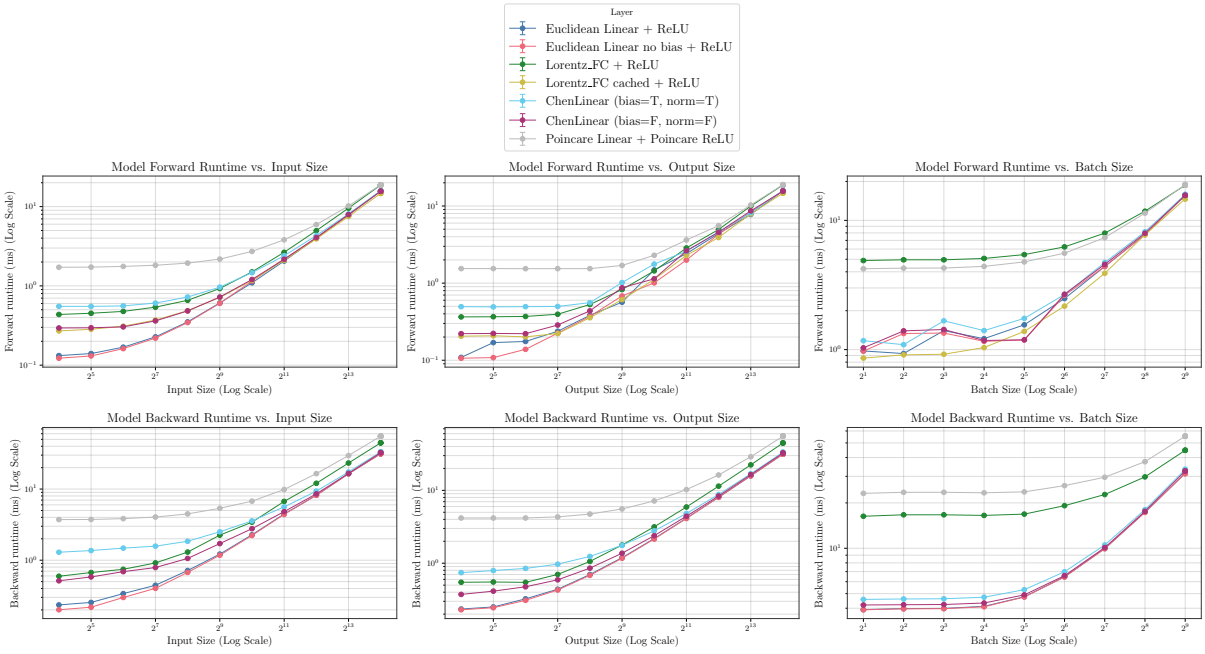
Lorentz versions: When comparing to MLRs, we benchmark the MLR introduced by Bdeir et al., 2023. When compared to FC, we benchmark the formulation by Chen et al., 2022 with the normalization and with bias, and without.

Poincare: When comparing to MLRs, we benchmark the MLR from Shimizu et al., 2021. When compared to FC, we benchmark the fully connected version of Shimizu et al., 2021 with the activation function ReLU applied in the tangent space introduced by van Spengler et al., 2023.

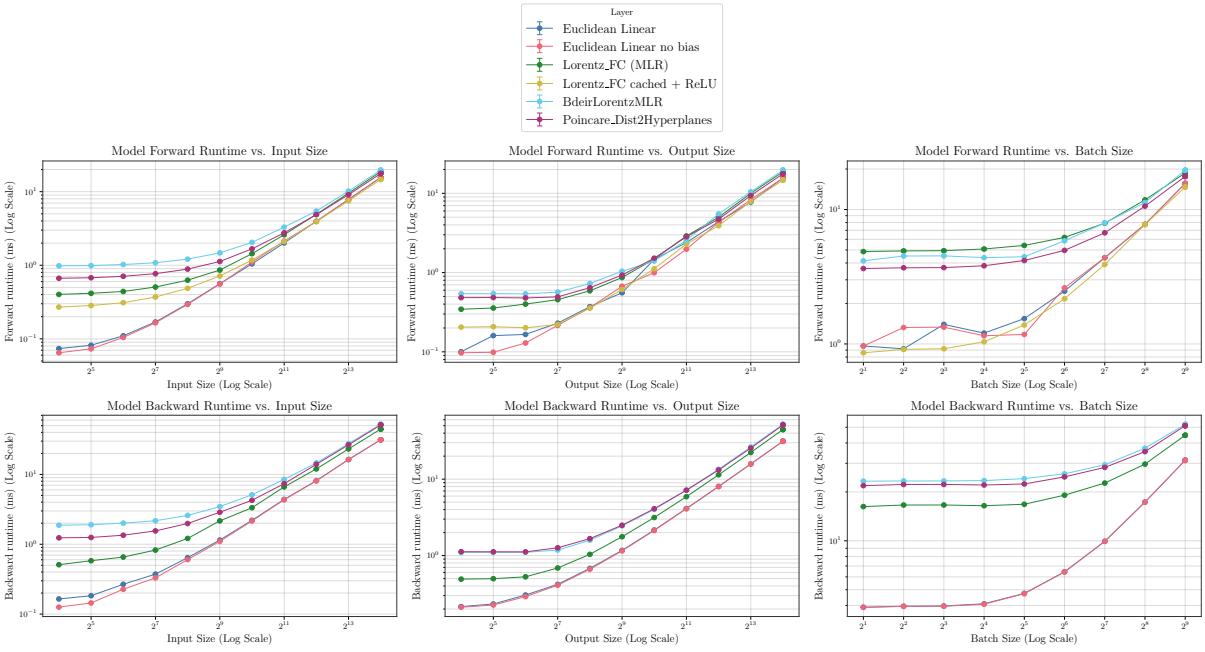
5.1.2 Results

The results can be seen in the Figures 5.1 and A.1. Analyzing them we see that the fastest layers in almost every run on low dimensions are the Euclidean layers with and without bias. In those settings, the cached version of our Lorentz FC and MLR is performing almost as good as Euclidean, scaling better with the batch size. In the high-dimensional range, both our cached version of Lorentz FC and MLR, and the Euclidean without bias take the lead, surpassing in computational efficiency the Euclidean with bias. The reason is that the cached version and the Euclidean without bias only perform matrix multiplications and multiplication (and in FC activation function), and, although the dimensions of the matrix multiplications are bigger by one, it is more efficient than adding the bias vector.

Comparing the non cached versions of our Lorentz model with the other hyperbolic layers, it consistently performs between Chen's Lorentz version without normalization or bias, and the version with normaliation and bias.



(a)



(b)

Figure 5.1: Runtime comparisons for different types of layers. **(a) Fully connected layers**. **(b) MLR layers**. Runtime of forward (upper row) and backward (lower row) pass of fully connected layers averaged over 50 iterations vs. the size of the input (left column), output (middle column), and batch (right column) sizes. Per plot, the other two dimensions of input, output, and batch size that are not compared are fixed to values: input: 2^{14}, output: 2^{14}, batch: 2^9. Notice that in lower dimensions the Euclidean layers are the fastest, followed by the cached version of the Lorentz fully connected, but at high dimensions the cached Lorentz fully connected is faster than the one with bias. The axes are in log scale, find the same plots in linear scale in Appendix Figure A.1, where you can see better the differences in the high dimensions.

5.2 Experiment 2: Fine-grained Image Classification

In this experiment we train and evaluate the classification models on standard fine-grained visual classification (FGVC) benchmarks. This task is chosen to assess the representational capacity of the different hyperbolic heads in a realistic setting.

5.2.1 Setup

Datasets: Our experiments are conducted on three standard FGVC benchmarks. For each dataset, images are pre-processed and split into distinct training, validation, and test sets.

FGVC-Aircraft: This dataset consists of 10,000 images of aircraft, categorized into 100 classes corresponding to different model variants. The data is partitioned into a training set of 6,667 images and a test set of 3,333 images. We use the standard splits provided with the dataset (Maji et al., 2013).

Oxford Flowers-102: This dataset contains 8,189 images of flowers from 102 different species common to the United Kingdom. The standard split provides 2,040 images for training and 6,149 for testing. We further divide the official training set into a new training set of 1,020 images and a validation set of 1,020 images (Nilsback & Zisserman, 2008).

Stanford Dogs: This dataset features 20,580 images of 120 breeds of dogs from around the world. The official splits provide 12,000 training images and 8,580 test images. To create a validation set, we perform a stratified partition on the official training data, assigning 80% (9,600 images) to our training set and 20% (2,400 images) to our validation set (Khosla et al., 2011).

Metrics: We report the following metrics:

- **Top-1 and Top-5 Accuracy:** Percentage of predictions where the correct class ranks among the top 1 or 5 highest-probability predictions.
- **Averaged Precision, Recall, and F1-Score**: We compute those metrics per class, and report the averaged over all classes.

Model Architecture: All models follow a two-stage design consisting of a fixed backbone that works as a feature extractor and a trainable classification head.

We use a ResNet-50 model (He et al., 2015), pre-trained on ImageNet-1K, as a frozen feature extractor. The final classification layer is removed, and a global average pooling operation is applied to the last feature map to produce a 2048-dimensional feature vector for each image. The backbone's weights remain frozen during all experiments.

Classification Heads: The extracted 2048-dimensional feature vectors are passed through the corresponding exponential map, and that serves as input to the following classification heads, which are the only components trained in our pipeline:

- `Baseline_Euclidean`: Linear layer with bias.
- `Lorentz_fully_connected_forward`: This is the layer we defined in the previous chapter. To make the classification, we map to a # of classes dimensional Lorentzian space, and we take only the space components.
- `Lorentz_fully_connected_mlr_angle`: It is parametrized as our defined layer, but the forward is just applying the signed scaled distance to the hyperplane.
- `Lorentz_fully_connected_mlr_dist`: It is the same as the last layer, but instead of scaling the angle of the Lorentzian product, we scale the total distance to the hyperplanes.
- `BdeirLorentzMLR_model`: The Lorentzian MLR described in Bdeir et al., 2023. It is equivalent to `Lorentz_fully_connected_mlr_dist`, but the hyperplanes are at a scaled distance from the origin.
- `Poincare_MLR`: The Poincaré MLR described in Shimizu et al., 2021.

5.2.2 Training Protocol

The training process is the same for every classification head. We apply the same transformations to the input data, afterwards, we do hyperparameter optimization, maximizing validation accuracy. And then we train with those hyperparameters until no improvement in the validation accuracy for 7 epochs. We repeat the training process 10 times and report average metrics.

Offline Feature Extraction and Augmentation: Instead of augmenting images on the fly, we generate feature sets offline that will be used by every classification head to accelerate the training loop. Before that, we compute for each dataset mean and standard deviation values for each RGB channel and normalize the images before extracting the features with the backbone.

Hyperparameter Optimization: We use the Optuna library (Akiba et al., 2019) to find the optimal hyperparameters for each model on each dataset. To do that, we run a search for 50 trials optimizing the validation accuracy. The search space and priors for the next hyperparameters are:

- **Learning Rate:** Log-uniform distribution between 10^{-5} and 10^{-2}.
- **Weight Decay:** Log-uniform distribution between 10^{-6} and 10^{-3}.
- **Batch Size:** Categorical choice from $\{64, 128, 256, 512\}$.
- **Curvature (κ):** Log-uniform distribution between 10^{-5} and 3.0 for hyperbolic models.

To select hyperparameter candidates for each trial, we use the TPESampler (Bergstra et al., 2011), and to prune unsuccessful runs, we use the median pruner.

Final Training: With the best hyperparameters identified, we train the final models. We use AdamW optimizer, Cross-entropy loss, and early stopping with 10 epochs of patience for the validation accuracy. For the epoch with the highest validation accuracy, we do a run in the test set to compute metrics. We repeat this training process 10 times to have a more robust estimation of the performance of the models, reporting average metrics and standard deviation over these 10 runs.

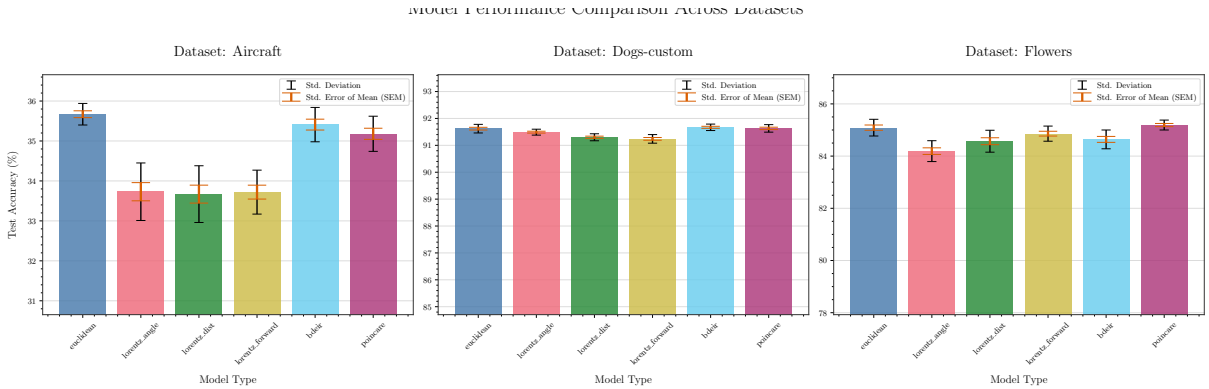


Figure 5.2: Test accuracy in fine grained classification datasets averaged over 10 runs. The black error bars are the standard deviation of the 10 runs, the red bars are the estimated standard deviation of the mean.

5.2.3 Results

The results along all three datasets, position our lorentz layers last, they are very slightly underperforming, however, our estimate is robust, as we repeated the training several times.

However, the selected curvatures by our automatic hyperparameter tuning pipeline for all of the hyperbolic models were quite small in magnitude. That, and the Euclidean model being the best model or on par with the best might indicate, that this specific setting is not the most favorable for testing hyperbolic classification heads. It is however noticeable, that our Lorentz models do not only achieve less performance than the euclidean, but than the other hyperbolic models too.

Table 5.1: Test performance on the **FGVC-Aircraft** dataset. Best scores per metric in **bold**.

Model	Acc (%)	Top-5 Acc (%)	Precision	Recall	F1
Euclidean	**35.67(0.27)**	61.91(0.30)	**0.37(0.00)**	**0.36(0.00)**	**0.36(0.00)**
Lorentz Angle	33.73(0.72)	59.65(0.55)	**0.37(0.01)**	0.34(0.01)	0.33(0.01)
Lorentz Dist	33.67(0.71)	59.66(0.48)	0.36(0.01)	0.34(0.01)	0.33(0.01)
Lorentz Forward	33.72(0.55)	59.59(0.60)	0.36(0.01)	0.34(0.01)	0.34(0.01)
BDEIR	35.41(0.43)	**61.97(0.35)**	**0.37(0.00)**	0.35(0.00)	0.35(0.00)
Poincaré	35.18(0.44)	61.37(0.59)	**0.37(0.01)**	0.35(0.00)	0.35(0.00)

Table 5.2: Test performance on the **Stanford Dogs** dataset. Best scores per metric in **bold**.

Model	Acc (%)	Top-5 Acc (%)	Precision	Recall	F1
Euclidean	91.62(0.16)	**99.55(0.03)**	0.92(0.00)	0.92(0.00)	0.92(0.00)
Lorentz Angle	91.49(0.11)	99.50(0.04)	0.92(0.00)	0.91(0.00)	0.91(0.00)
Lorentz Dist	91.30(0.13)	99.45(0.06)	0.92(0.00)	0.91(0.00)	0.91(0.00)
Lorentz Forward	91.24(0.16)	99.44(0.03)	0.92(0.00)	0.91(0.00)	0.91(0.00)
BDEIR	**91.67(0.12)**	99.52(0.02)	0.92(0.00)	**0.92(0.00)**	**0.92(0.00)**
Poincaré	91.63(0.14)	99.51(0.04)	**0.92(0.00)**	**0.92(0.00)**	**0.92(0.00)**

Table 5.3: Test performance on the **Oxford Flowers-102** dataset. Best scores per metric in **bold**.

Model	Acc (%)	Top-5 Acc (%)	Precision	Recall	F1
Euclidean	85.09(0.32)	95.69(0.21)	**0.87(0.00)**	0.85(0.00)	0.85(0.00)
Lorentz Angle	84.19(0.40)	95.22(0.22)	0.86(0.00)	0.84(0.00)	0.84(0.00)
Lorentz Dist	84.57(0.42)	95.37(0.24)	0.86(0.00)	0.85(0.00)	0.85(0.00)
Lorentz Forward	84.86(0.29)	95.62(0.07)	0.86(0.00)	0.85(0.00)	0.85(0.00)
BDEIR	84.64(0.36)	95.49(0.23)	0.86(0.00)	0.85(0.00)	0.85(0.00)
Poincaré	**85.19(0.19)**	**95.69(0.11)**	**0.87(0.00)**	**0.85(0.00)**	**0.85(0.00)**

Discussion 6

Most deep learning architectures implicitly make the decision to assign latent space in which data representations evolve an Euclidean structure. This is a significant decision, one that is rarely questioned. If we consider hyperbolic and Euclidean spaces as equivalent when curvature approaches zero, it seems rather arbitrary to treat zero curvature as the universally optimal value for every task. Reframing curvature as a learnable or tunable parameter raises a critical question: Why is Euclidean space so often assumed to be the best possible choice?

This preference appears to be driven less by theoretical justification and more by practical convenience. Euclidean layers are easier to interpret, more widely implemented, and better supported by existing libraries and documentation. In contrast, hyperbolic models remain relatively scarce, fragmented across inconsistent methodologies, and significantly slower in both training and inference. These factors contribute to a bias in favor of Euclidean architectures, not necessarily because they are optimal, but because they are more accessible.

We conclude the thesis by reflecting on some key insights and implications of this thesis. We discuss the contributions and outline possible directions for future research.

6.1 Contributions

The central contribution of this work is the formulation of a hyperplane based fully connected layer in the Lorentz model, benefiting from the previous research on the Poincaré model with the computational advantage of the Lorentz model. By formulating the signed, scaled distance to a hyperplane in Lorentzian space, we created a common mathematical foundation that supports both comparison and practical implementation. This led to the development of an efficient Lorentzian fully connected layer and a new class of activation functions that simplify operations, significantly increasing the speed.

Some important findings:

1. **A Computationally Efficient and Geometrically Principled Layer:** The Lorentzian fully connected layer, introduced in our work, offers significant runtime benefits. As demonstrated in Section 5.1, the use of Lorentzian activations like ReLU eliminates the need for costly arcsinh and sinh operations, reducing the forward pass to a simple matrix multiplication. Additionally, with the strategy of caching the intermediate matrix V (Section 4.3.2), we achieve a very significant speedup over Poincaré-based networks and even outperform Euclidean layers with bias in high-dimensional regimes during inference. This caching strategy does not compromise the model size or the possibility of alternating training and inference stages due to the invertibility of the mapping from parameters to

V. This addresses a major barrier to the adoption of hyperbolic models: computational efficiency.

2. **The Inductive Bias of Hyperbolic Geometry is Not a Universal Advantage:** As shown in our fine-grained visual classification experiments (Section 5.2), Lorentzian models did not consistently outperform either Euclidean or other hyperbolic baselines. Interestingly, hyperparameter optimization consistently yielded small curvature values (κ), pushing the learned geometry toward Euclidean space, and none of the other hyperbolic methods significantly surpassed Euclidean. This might be an indication that, the task performed is not best suitable for hyperbolic classifiers. Learning from the features generated by the Euclidean ResNet may not map better in hyperbolic space than in Euclidean one.
3. **Unification Clarifies Design Choices:** By deriving our fully connected layer from first principles, we clarify and generalize several ad-hoc approaches in prior work. For example, applying a standard linear transformation to ambient Lorentzian coordinates followed by projection (Section 3.2) is shown to be a special case of our broader framework. Our analysis illustrates why temporal and spatial components must be treated differently, and why hyperplane normals $\mathbf{w}$ must be constrained to be space-like. This moves model design from empirical heuristics to principled, geometry-principled construction in the Lorentz model.

6.2 Limitations and Future Work

While this thesis makes a substantial contribution, several limitations and open questions remain:

1. **Fully Hyperbolic Deep Learning:** Our experiments relied on frozen Euclidean backbones. This might have limited the scope of our experiments. Incorporating the Lorentzian fully connected layer into existing methods to perform fully hyperbolic deep learning is a natural step.
2. **Exploring Truly Hyperbolic Datasets:** Future work should evaluate our layers on tasks with a known hierarchical structure, like graph representation learning or natural language processing.
3. **Advanced Optimization Techniques:** We parameterized the weight matrix V using learnable Euclidean parameters $(U, \mathbf{b})$, which may not be ideal for navigating the curved parameter space as there is some error associated with the discretization when representing U that will be passed when mapping to V. As suggested in Section 4.3.2, future work could involve directly learning V on the Riemannian manifold of spacelike vectors using Riemannian Stochastic Gradient Descent (RSGD).

6.3 Conclusion

This thesis has translated the Poincaré geometrically principled Fully connected layer to the Lorentz model. Our derivation of the Lorentzian fully connected layer not only clarified prior methods but also produced

a formulation with competitive computational performance compared to Euclidean layers.

While our empirical classification results suggest a slight underperformance when learning with the Lorentz Fully connected, we provide an elementary block for further research and development in hyperbolic neural networks. Ultimately, this work brings us closer to a future in which the choice of geometry is driven not by convenience, but by the fundamental structure of the problem.

Appendix

Additional Proofs and Results

A

Proposition A.0.1 *In the limit where the curvature $\kappa \to 0^+$, the following two expressions for the signed distance tend to the same value:*

$$d_1 = \frac{\|\mathbf{w}\|_{\mathscr{L}}}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\frac{\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})}{\|\mathbf{w}\|_{\mathscr{L}}}\right) \tag{A.1}$$

$$d_2 = \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})\right) \tag{A.2}$$

Specifically, both converge to the Euclidean inner product $\mathbf{x} \circ \mathbf{w}$.

Proof. To prove this, we evaluate the limit of each expression using the fundamental limit identity, which can be verified with L'Hôpital's Rule:

$$\lim_{u \to 0} \frac{\operatorname{arcsinh}(u)}{u} = 1$$

For the first expression, we rearrange the terms and apply the limit:

$$\lim_{\kappa \to 0^+} d_1 = \lim_{\kappa \to 0^+} \left(\frac{\|\mathbf{w}\|_{\mathscr{L}}}{\sqrt{\kappa}} \cdot \frac{\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})}{\|\mathbf{w}\|_{\mathscr{L}}}\right) \cdot \left(\frac{\operatorname{arcsinh}\left(\frac{\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})}{\|\mathbf{w}\|_{\mathscr{L}}}\right)}{\frac{\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})}{\|\mathbf{w}\|_{\mathscr{L}}}}\right)$$

$$= (\mathbf{x} \circ \mathbf{w}) \cdot \lim_{u \to 0} \frac{\operatorname{arcsinh}(u)}{u} \quad \text{where } u = \frac{\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})}{\|\mathbf{w}\|_{\mathscr{L}}}$$

$$= \mathbf{x} \circ \mathbf{w}$$

For the second expression, we perform a similar rearrangement:

$$\lim_{\kappa \to 0^+} d_2 = \lim_{\kappa \to 0^+} \left(\frac{1}{\sqrt{\kappa}} \cdot \sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})\right) \cdot \left(\frac{\operatorname{arcsinh}(\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w}))}{\sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})}\right)$$

$$= (\mathbf{x} \circ \mathbf{w}) \cdot \lim_{v \to 0} \frac{\operatorname{arcsinh}(v)}{v} \quad \text{where } v = \sqrt{\kappa}(\mathbf{x} \circ \mathbf{w})$$

$$= \mathbf{x} \circ \mathbf{w}$$

Since both expressions converge to $\mathbf{x} \circ \mathbf{w}$, the proposition is proven. □

Proof of Parameter Recovery from w

Proposition A.0.2 *Let* $\mathbf{w}$ *be space-like normal vector* $\mathbf{w} = (w_1, \overline{\mathbf{w}})$.

Our starting point is the set of equations from Section 4.2 that define $\mathbf{w}$ *in terms of* $\mathbf{u} \in \mathbb{R}^n$ *and* $b \in \mathbb{R}$*:*

$$w_1 = \|\mathbf{u}\|_E \sinh(\sqrt{\kappa} b \|\mathbf{u}\|_E) \tag{A.3}$$

$$\overline{\mathbf{w}} = \cosh(\sqrt{\kappa} b \|\mathbf{u}\|_E)\mathbf{u} \tag{A.4}$$

Our goal is to recover $\mathbf{u}$ *and* b *as a function of* $\mathbf{w}$*. We are going to prove this is:*

$$\mathbf{u} = \frac{\overline{\mathbf{w}}}{\|\overline{\mathbf{w}}\|_E} \|\mathbf{w}\|_{\mathcal{L}} \tag{A.5}$$

$$b = \frac{1}{\sqrt{\kappa}\|\mathbf{w}\|_{\mathcal{L}}} \operatorname{arcsinh}\left(\frac{w_1}{\|\mathbf{w}\|_{\mathcal{L}}}\right) \tag{A.6}$$

Derivation of u By construction

$$\|\mathbf{u}\|_E = \|\mathbf{w}\|_{\mathcal{L}} \tag{A.7}$$

$$\mathbf{u} \propto \overline{\mathbf{w}} \tag{A.8}$$

Therefore:

$$\mathbf{u} = \frac{\overline{\mathbf{w}}}{\|\overline{\mathbf{w}}\|_E} \|\mathbf{w}\|_{\mathcal{L}} \tag{A.9}$$

Derivation of b The parameter b is related to the signed distance from the origin $\mathbf{0}_{\mathbb{L}_\kappa^n}$ to the hyperplane $H_{\mathbf{w}}$. By construction, this distance is $d_{\mathbb{L}}^{\text{signed}}(\mathbf{0}_{\mathbb{L}_\kappa^n}, H_{\mathbf{w}}) = b\|\mathbf{u}\|_E$.

$$b\|\mathbf{u}\|_E = d_{\mathbb{L}}^{\text{signed}}(\mathbf{0}_{\mathbb{L}_\kappa^n}, H_{\mathbf{w}}) = \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\frac{\sqrt{\kappa}(\mathbf{0}_{\mathbb{L}_\kappa^n} \circ \mathbf{w})}{\|\mathbf{w}\|_{\mathcal{L}}}\right) \tag{A.10}$$

The Lorentzian inner product of the origin $\mathbf{0}_{\mathbb{L}_\kappa^n} = (1/\sqrt{\kappa}, \mathbf{0})$ and $\mathbf{w}$ is $\mathbf{0}_{\mathbb{L}_\kappa^n} \circ \mathbf{w} = -w_1/\sqrt{\kappa}$. Substituting this into the distance formula gives:

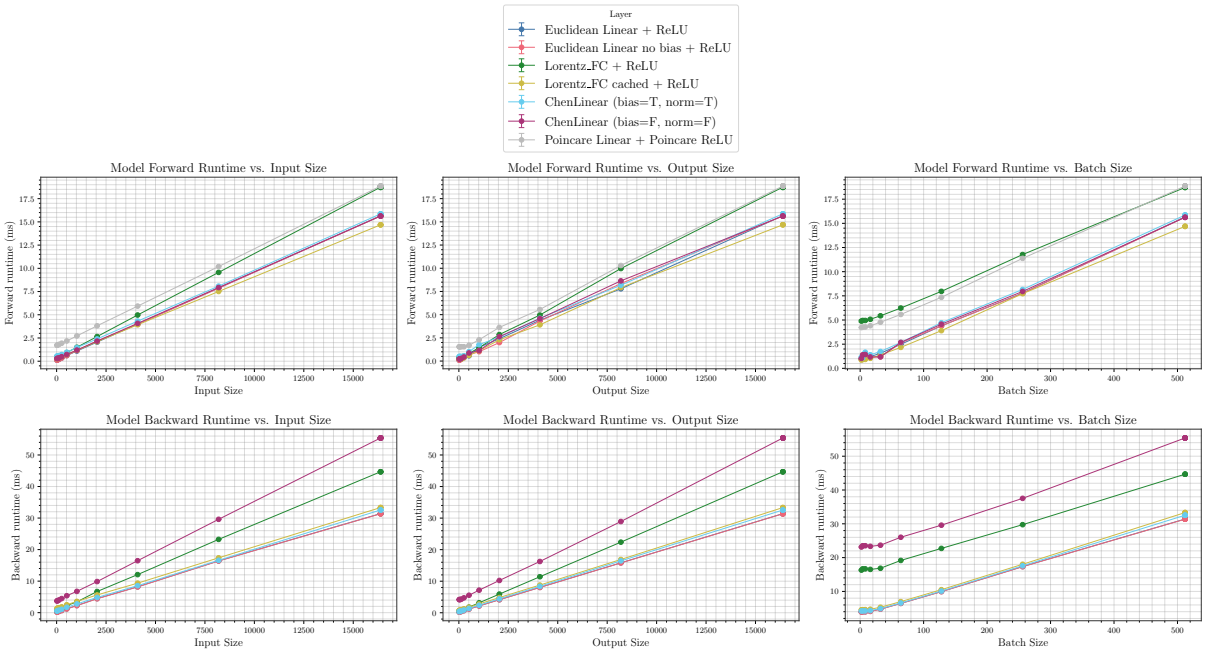
$$d_{\mathbb{L}}^{\text{signed}}(\mathbf{0}_{\mathbb{L}_\kappa^n}, H_{\mathbf{w}}) = \frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\frac{\sqrt{\kappa}(-w_1/\sqrt{\kappa})}{\|\mathbf{w}\|_{\mathcal{L}}}\right) = -\frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\frac{w_1}{\|\mathbf{w}\|_{\mathcal{L}}}\right) \tag{A.11}$$

Equating the two expressions for the signed distance:

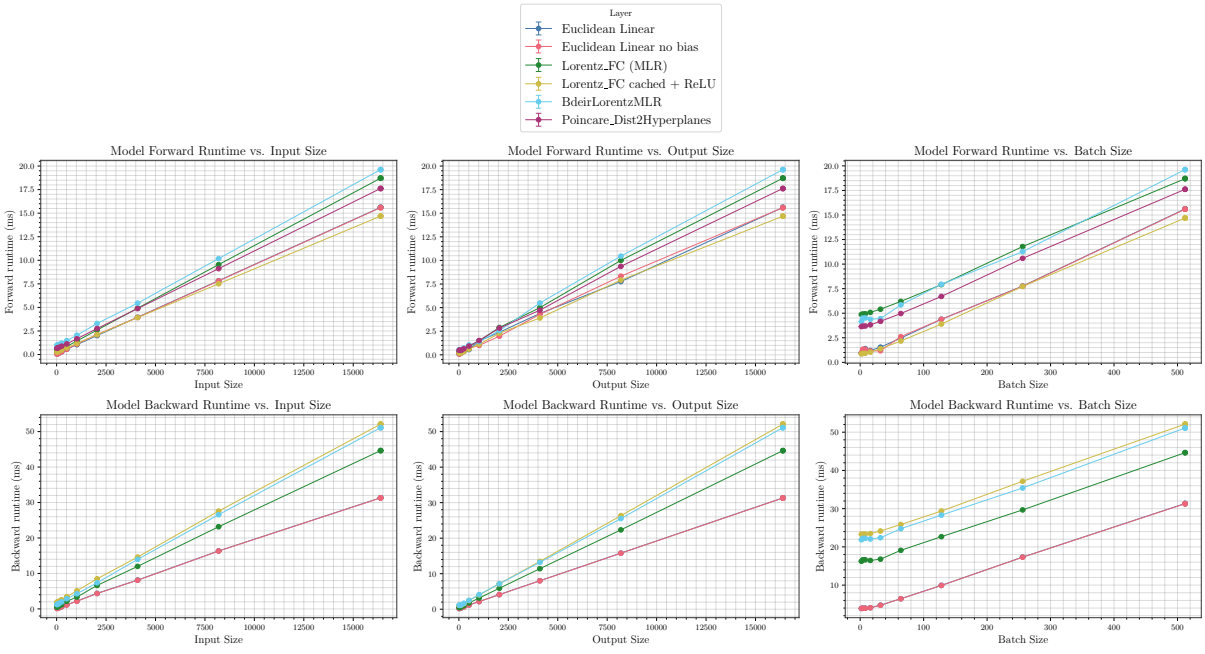
$$b\|\mathbf{u}\|_E = -\frac{1}{\sqrt{\kappa}} \operatorname{arcsinh}\left(\frac{w_1}{\|\mathbf{w}\|_{\mathcal{L}}}\right) \tag{A.12}$$

Finally, isolating b and substituting $\|\mathbf{u}\|_E = \|\mathbf{w}\|_{\mathcal{L}}$ yields the final expression:

$$b = -\frac{1}{\sqrt{\kappa}\|\mathbf{u}\|_E} \operatorname{arcsinh}\left(\frac{w_1}{\|\mathbf{w}\|_{\mathcal{L}}}\right) = -\frac{1}{\sqrt{\kappa}\|\mathbf{w}\|_{\mathcal{L}}} \operatorname{arcsinh}\left(\frac{w_1}{\|\mathbf{w}\|_{\mathcal{L}}}\right) \tag{A.13}$$



(a)



(b)

Figure A.1: Runtime comparisons for different types of layers in **linear scale**. **(a) Fully connected layers**. **(b) MLR layers**. Runtime of forward (upper row) and backward (lower row) pass of fully connected layers averaged over 50 iterations vs. the size of the input (left column), output (middle column) and batch (right column) sizes. Per plot, the other two dimensions of input, output and batch size that are not compared are fixed to values: input: 2^{14}, output: 2^{14}, batch: 2^9. Notice that in lower dimensions the Euclidean layers are the fastest, followed by the cached version of the Lorentz fully connected. At high dimensions, the cached Lorentz fully connected is faster than the version with bias. These linear plots allow to see clearly the differences at higher values.

Bibliography

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Ayubcha, C., Sajed, S., Omara, C., Veldman, A. B., Singh, S. B., Lokesha, Y. U., Liu, A., Aziz-Sultan, M. A., Smith, T. R., & Beam, A. (2024). Improved generalizability in medical computer vision: Hyperbolic deep learning in multi-modality neuroimaging. *Journal of Imaging*, *10*. https://doi.org/10.3390/jimaging10120319

Bdeir, A., Schwethelm, K., & Landwehr, N. (2023). Fully Hyperbolic Convolutional Neural Networks for Computer Vision. *12th International Conference on Learning Representations, ICLR 2024*.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Proceedings of the 25th International Conference on Neural Information Processing Systems*, 2546–2554.

Chami, I., Ying, R., Ré, C., & Leskovec, J. (2019). Hyperbolic Graph Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, *32*.

Chavel, I. (2006). *Riemannian geometry: A modern introduction* (2nd). Cambridge University Press.

Chen, W., Han, X., Lin, Y., Zhao, H., Liu, Z., Li, P., Sun, M., & Zhou, J. (2022, May). Fully hyperbolic neural networks. In S. Muresan, P. Nakov, & A. Villavicencio (Eds.), *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 5672–5686). Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.acl-long.389

Ganea, O., Becigneul, G., & Hofmann, T. (2018). Hyperbolic neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 31). Curran Associates, Inc.

Gulcehre, C., Denil, M., Malinowski, M., Razavi, A., Pascanu, R., Hermann, K. M., Battaglia, P., Bapst, V., Raposo, D., Santoro, A., & De Freitas, N. (2018). Hyperbolic Attention Networks. *7th International Conference on Learning Representations, ICLR 2019*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Khosla, A., Jayadevaprakash, N., Yao, B., & Fei-Fei, L. (2011). Novel dataset for fine-grained image categorization. *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*.

Maji, S., Kannala, J., Rahtu, E., Blaschko, M., & Vedaldi, A. (2013). *Fine-grained visual classification of aircraft* (tech. rep.).

Mettes, P., Ghadimi Atigh, M., Keller-Ressel, M., Gu, J., & Yeung, S. (2024). Hyperbolic deep learning in computer vision: A survey. *Int. J. Comput. Vision*, *132*(9), 3484–3508. https://doi.org/10.1007/s11263-024-02043-5

Mishne, G., Wan, Z., Wang, Y., & Yang, S. (2023, 23–29 Jul). The numerical stability of hyperbolic representation learning. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, & J. Scarlett (Eds.), *Proceedings of the 40th international conference on machine learning* (pp. 24925–24949, Vol. 202). PMLR.

Nilsback, M.-E., & Zisserman, A. (2008). Automated flower classification over a large number of classes. *Indian Conference on Computer Vision, Graphics and Image Processing*.

Peng, W., Varanka, T., Mostafa, A., Shi, H., & Zhao, G. (2022). Hyperbolic deep neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(12), 10023–10044. https://doi.org/10.1109/TPAMI.2021.3136921

Ratcliffe, J. G. (2019). *Foundations of hyperbolic manifolds* (3rd ed., Vol. 149). Springer Cham.

Shimizu, R., Mukuta, Y., & Harada, T. (2021). Hyperbolic neural networks++. *International Conference on Learning Representations*.

van Spengler, M., Berkhout, E., & Mettes, P. (2023). Poincare ResNet. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 5419–5428.

Yang, M., Zhou, M., Li, Z., Liu, J., Pan, L., Xiong, H., & King, I. (2022). Hyperbolic Graph Neural Networks: A Review of Methods and Applications.