CONSTANT CURVATURE GRAPH CONVOLUTIONAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Interest has been rising lately towards methods representing data in non-Euclidean spaces, e.g. hyperbolic or spherical. These geometries provide specific inductive biases useful for certain real-world data properties, e.g. scale-free or hierarchical graphs are best embedded in a hyperbolic space. However, the very popular class of *graph neural networks* is currently limited to model data only via Euclidean node embeddings and associated vector space operations. In this work, we bridge this gap by proposing mathematically grounded generalizations of *graph convolutional networks* (GCN) to (products of) constant curvature spaces. We do this by i) extending the gyro-vector space theory from hyperbolic to spherical spaces, providing a unified and smooth view of the two geometries, ii) leveraging gyro-barycentric coordinates that generalize the classic Euclidean concept of the *center of mass*. Our class of models gives strict generalizations in the sense that they recover their Euclidean counterparts when the curvature goes to zero from either side. Empirically, our methods outperform different types of classic Euclidean GCNs in the tasks of node classification and minimizing distortion for symbolic data exhibiting non-Euclidean behavior, according to their discrete curvature.

1 INTRODUCTION

Euclidean geometry in ML. In machine learning (ML), data is most often represented in a Euclidean space for various reasons. First, some data is *intrinsically* Euclidean, such as positions in 3D space in classical mechanics. Second, intuition is easier in such spaces, as they possess an appealing vectorial structure allowing basic arithmetic and a rich theory of linear algebra. Finally, a lot of quantities of interest such as distances and inner-products are known in closed-form formulae and can be computed very efficiently on the existing hardware. These operations are the basic building blocks for most of today's popular machine learning models. Thus, the powerful simplicity and efficiency of Euclidean geometry has led to numerous methods achieving state-of-the-art on tasks as diverse as machine translation Bahdanau et al. (2014); Vaswani et al. (2017), speech recognition Graves et al. (2013), image classification He et al. (2016) or recommender systems He et al. (2017).

Riemannian ML. In spite of this success, certain types of data have been shown to be better represented by other types of geometries Bronstein et al. (2017); Nickel & Kiela (2017), leading in particular to the rich theories of manifold learning Roweis & Saul (2000); Tenenbaum et al. (2000) and information geometry Amari & Nagaoka (2007). The mathematical framework in vigor to manipulate non-Euclidean geometries is known as *Riemannian geometry* Spivak (1979). We briefly discuss some concepts of this machinery in appendix B. Although its theory leads to many strong and elegant results, some of its basic quantities such as the distance function $d(\cdot, \cdot)$ are in general not available in closed-form, which can be prohibitive to many computed-based methods.

Geometries of Constant Curvature. An interesting trade-off between general Riemannian manifolds and the Euclidean space is given by manifolds of *constant sectional curvature*. They define together what are called *hyperbolic geometry* (negative curvature), *elliptic geometry* (positive curvature) and Euclidean geometry (zero curvature). One of their advantage is that most of the quantities of interest for machine learning methods are known in closed-form formulae: distance, geodesics, exponential map, parallel transport, as well as their gradients. These geometries also possess their own partial analogues to the Euclidean vector space formalism, with formulae for weighted centroids such as barycentric coordinates.

The **hyperbolic space** can be intuitively understood as a continuous tree: the volume of a ball grows exponentially with its radius, similarly as how the number of nodes in a binary tree grows exponentially with its depth. Its tree-likeness properties have long been studied mathematically Gromov (1987); Hamann (2017); Ungar (2008), were related to heterogeneous properties of complex networks Krioukov et al. (2010) and used to visualize large hierarchies Lamping et al. (1995). Recently, in machine learning, hyperbolic representations outperformed their Euclidean analogues in a number of embedding and classification tasks Cho et al. (2019); De Sa et al. (2018); Ganea et al. (2018a); Gu et al. (2019); Nickel & Kiela (2018; 2017); Tifrea et al. (2019). Several important tools or methods found their hyperbolic counterparts, such as variational autoencoders Mathieu et al. (2019); Ovinnikov (2019), attention mechanisms Gulcehre et al. (2018), matrix multiplications, recurrent units and multinomial logistic regression Ganea et al. (2018b).

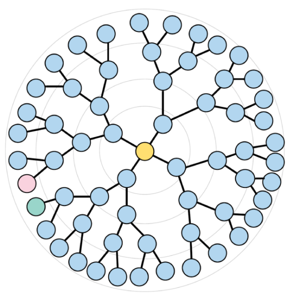
Figure 1: Euclidean space quickly "runs out of space" when fitting exponentially volume growing data such as trees. This issue is formalised in appendix C.

Similarly, **spherical geometry** provides benefits for modeling certain types of data Matousek (2013); Davidson et al. (2018); Xu & Durrett (2018); Gu et al. (2019); Grattarola et al. (2018); Wilson et al. (2014).

Graph Embeddings. The Euclidean space has certain limitations of embedding various types of graphs without large distortion. In some of these cases (e.g. scale-free graphs and complex networks Krioukov et al. (2010)), the hyperbolic and spherical spaces do not suffer from this representational bottleneck and can provide a better inductive bias for the specific data types. We detail this discussion in appendix C.

"Linear Algebra" of Constant Curvature Spaces – Gyrovector Spaces. The hyperbolic space is especially useful in Einstein's special relativity theory where particles' velocities are mathematically represented as vectors in the Poincaré ball model. In order to study this geometry in analogy with the vector space formalism of the Euclidean geometry, Ungar (1999; 2005; 2008; 2016) proposed the elegant non-associative algebraic formalism of **gyrovector spaces** [1]. Relativistic composition of velocities now becomes the *gryo-addition*. Recently, Ganea et al. (2018b) have generalized matrix multiplication and pointwise non-linearities to gyro spaces, allowing them to stack these building blocks for generalizing classic deep learning models to operate with hyperbolic data representations.

However, *it remains unclear how to extend in a principled manner the gyrovector space theory to spaces of constant positive curvature (spherical).* By leveraging Euler's formula and complex analysis, as far as we know, we present the first unified gyro framework that smoothly interpolates between geometries of constant curvatures irrespective of their signs. Tools especially important for machine learning such as exponential map, geodesics, parallel transport, distance, barycentric coordinates have efficient closed form expressions in these spaces and converge to their Euclidean variants when the curvature vanishes from both positive and negative regimes. This further allows us to develop deep learning architectures in products of constant curvature spaces.

Graph Neural Networks. Kipf & Welling (2016) recently proposed a scalable approach to train deep network models directly on graphs, sharing parameters in a manner that is consistent with the geometry of the graph. As the weight sharing scheme is inspired from graph convolutions, involving the graph Laplacian, they were coined *graph convolutional networks* (GCN). The proposed architecture is essentially made of interpolating node embeddings via a (symmetrically) normalized adjacency matrix, while this weight sharing can be understood as an efficient diffusion-like regularizer. Similar works Abu-El-Haija et al. (2018); Hamilton et al. (2017); Defferrard et al. (2016); Klicpera et al. (2019); Veličković et al. (2018) showed success at a number of tasks including link prediction Zhang & Chen (2018), graph classification Xu et al. (2019) and node classification Kipf & Welling (2017); Klicpera et al. (2019); Veličković et al. (2018). We describe GCN and discuss some related work in more detail in appendix A, but a broader overview is given by Wu et al. (2019).

[1] `https://en.wikipedia.org/wiki/Gyrovector_space`

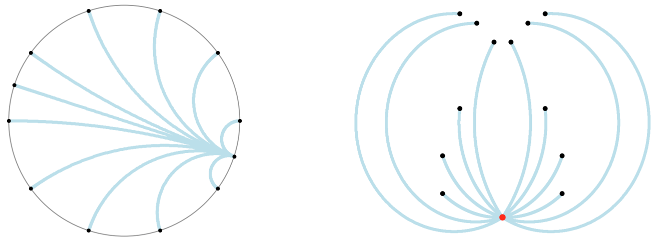

Figure 2: Geodesics in the Poincaré disk (left) and the stereographic projection of the sphere (right).

How should one adapt graph neural networks to non-flat geometries of constant curvature?

In this work, we propose to let GCN manipulate data via node embeddings lying in spaces of constant curvature or product of those, instead of a Euclidean space, in order to combine the representational power of these geometries with the effectiveness of GCNs. *Our contributions are as follows*:

- We extend the gyro-vector space theory from hyperbolic to spherical spaces using complex analysis. This gives unified and computationally efficient expressions for important Riemannian geometric tools needed in machine learning, e.g. matrix multiplication and barycentric coordinates. Our models are smooth deformations of the Euclidean vector space machinery.
- We leverage this apparatus to develop mathematically grounded adaptations of GCN to (products of) spaces of constant curvature.
- We empirically show the benefit of our models on several real and synthetic node classification and distortion minimization benchmarks.

2 The Geometry of Constant Curvature Spaces

Riemannian Geometry. A manifold $\mathcal{M}$ of dimension d is a generalization to higher dimensions of the notion of surface, and is a space that locally *looks* like $\mathbb{R}^d$. At each point $x \in \mathcal{M}$, $\mathcal{M}$ can be associated a *tangent space* $T_x\mathcal{M}$, which is a vector space of dimension d that can be understood as a first order approximation of $\mathcal{M}$ around x. A *riemannian metric* g is given by an inner-product $g_x(\cdot, \cdot)$ at each tangent space $T_x\mathcal{M}$, g_x varying smoothly with x. A given g defines the *geometry* of $\mathcal{M}$, because it can be used to define the distance between x and y as the infimum of the lengths of smooth paths $\gamma : [0, 1] \to \mathcal{M}$ from x to y, where the length is defined as $\ell(\gamma) := \int_0^1 \sqrt{g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t))}\mathrm{d}t$. Under certain assumptions, a given g also defines *curvature*. We detail these concepts in appendix B.

The Poincaré model. We choose to work in the *Poincaré* model of the hyperbolic geometry because it can be continuously deformed to the Euclidean space. For a curvature $-c < 0$, the Poincaré model of dimension $d \geq 2$ is defined as $\mathbb{P}_c^d := \{x \in \mathbb{R}^d \mid c\|x\|_2 < 1\}$ equipped with its *Riemannian metric* $g_x^{\mathbb{P}} = \frac{4}{(1-c||x||^2)^2}\mathbf{I} = (\lambda_x^c)^2\mathbf{I}$. What is convenient however, is that the distance induced by g between $x, y \in \mathbb{P}_c^d$ is given by the formula:

$$d_{\mathbb{P}}^c(x, y) = \frac{1}{\sqrt{c}} \cosh^{-1}\left(1 + \frac{\frac{2}{c}||x-y||^2}{(\frac{1}{c} - ||x||^2)(\frac{1}{c} - ||y||^2)}\right)$$

Notice that here it holds that $d_{\mathbb{P}}^c(x, y) \xrightarrow{c \to 0} 2||x - y||$, so in the limit we recover Euclidean geometry as desired.

Gyrovector spaces and Riemannian geometry of the Poincaré ball. As discussed in section 1, the gryovector space formalism is used to generalize vector spaces to hyperbolic geometry. In addition, important quantities from Riemannian geometry can be rewritten in terms of the Möbius vector addition and scalar-vector multiplication Ganea et al. (2018b). We define and detail gyrovector spaces in appendix D.

For $x, y \in \mathbb{P}_c^d$, the **Möbius addition** is given by:

$$x \oplus_{\mathbb{M}}^c y = \frac{(1 + 2cx^Ty + c||y||^2)x + (1 - c||x||^2)y}{1 + 2cx^Ty + c^2||x||^2||y||^2} \in \mathbb{P}_c^d$$

For $s \in \mathbb{R}$ and $x \in \otimes^c_{\mathbb{M}}$, the **scalar multiplication** in the Poincaré model $\mathbb{P}^d_c$ is given by:

$$s \otimes^c_{\mathbb{M}} x = \frac{1}{\sqrt{c}} \tanh\left(s \cdot \tanh^{-1}(\sqrt{c}||x||)\right) \frac{x}{||x||} \in \mathbb{P}^d_c$$

One can show that Möbius addition and scalar multiplication in $\mathbb{P}^d_c$ form a gyrovector space Ungar (2008) which is detailed in appendix D. We can use the Möbius addition to rewrite the distance formula as $d^c_{\mathbb{P}}(x, y) = \frac{2}{\sqrt{c}} \tanh^{-1}(\sqrt{c}|| - x \oplus^c_{\mathbb{M}} y||)$. This model is smoothly deformed to the Euclidean model in the curvature limit as $d^c_{\mathbb{P}}(x, y) \xrightarrow{c \to 0} 2\|x - y\|$. For $x, y \in \mathbb{P}^d_c$, the geodesic $\gamma_{x \to y} : \mathbb{R} \to \mathbb{P}^d_c$, connecting x and y is given by

$$\gamma_{x \to y}(t) = x \oplus^c_{\mathbb{M}} \left(t \otimes^c_{\mathbb{M}} (-x \oplus^c_{\mathbb{M}} y)\right)$$

Finally, the **exponential** and **logarithmic map** are written in closed form Ganea et al. (2018b): For $x, y \in \mathbb{P}^d_c$ and $v \neq 0$, $y \neq x$, the exponential map $\exp_x : T_x\mathbb{P}^d_c \to \mathbb{P}^d_c$ and the log map $\log_x : \mathbb{P}^d_c \to T_x\mathbb{P}^d_c$ are given by

$$\exp^c_x(v) = x \oplus^c_{\mathbb{M}} \left(\tanh\left(\sqrt{c}\frac{\lambda^c_x ||v||}{2}\right) \frac{v}{||v||} \right); \log^c_x(y) = \frac{2}{\sqrt{c}\lambda^c_x} \tanh^{-1}(\sqrt{c}|| - x \oplus^c_{\mathbb{M}} y||) \frac{-x \oplus^c_{\mathbb{M}} y}{|| - x \oplus^c_{\mathbb{M}} y||}$$

Gyrovector spaces of the Spherical Spaces. The gyrovector space formalism was so far, to our knowledge, only described for hyperbolic spaces. *So, how can we extend it to spaces of constant positive curvature (spherical) K ?* It turns out that replacing c by $-K$ in the above equations of the Poincaré model suffices. However, that implies that all occurrences of $\sqrt{c}$ would be replaced by $i\sqrt{K}$ where i is the imaginary unit. This will still give only real functions if one uses Euler's formula $e^{ix} = \cos(x) + i \sin(x)$ and the resulting identities $\tan(ix) = i \tanh(x)$ and $\tan^{-1}(ix) = i \tanh^{-1}(x)$. In this case, all the above equations will recover the corresponding quantities from the space of constant curvature obtained by isometrically projecting the sphere $\mathbb{S}^d_c$ into $\mathbb{R}^d$, i.e. the stereographic projection of the sphere. Moreover, the gyrovector space definition list would be fully satisfied as well. We denote this gyrospace and the two resulting Möbius gyro operations by $(\mathbb{G} = \mathbb{R}^d, \oplus^c_{\mathbb{G}}, \otimes^c_{\mathbb{G}})$. We discuss this in detail in appendix D.

As a consequence, we have a unified formalism that interpolates smoothly between all three geometries of constant curvature.

3 Hyperbolic & Spherical GCNs

We start by introducing the methods upon which we build. For space reasons, we will here detail just our proposed hyperbolic GCN. However, generalizations to spherical and cartesian products of constant curvature spaces Gu et al. (2019) are straightforward given the previous discussion about universality of gyrovector spaces.

3.1 Graph Convolutional Networks

The problem of node classification on a graph has long been tackled with explicit regularization using the graph Laplacian Weston et al. (2012). Namely, for a directed graph with adjacency matrix $\mathbf{A}$, by adding the following term to the loss: $\sum_{i,j} \mathbf{A}_{ij} \|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|^2 = f(\mathbf{X})^T \mathbf{L} f(\mathbf{X})$, where $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the (unnormalized) graph Laplacian, $D_{ii} := \sum_k A_{ik}$ defines the (diagonal) degree matrix, f contains the trainable parameters of the model and $\mathbf{X} = (x^j_i)_{ij}$ the node features of the model. Such a regularization is expected to improve generalization if connected nodes in the graph tend to share labels; node i with feature vector $\mathbf{x}_i$ is represented as $f(\mathbf{x}_i)$ in a Euclidean space.

With the aim to obtain more scalable models, Kipf et al. propose to make this regularization implicit by incorporating it into what they call *graph convolutional networks* Kipf & Welling (2016), which they motivate as a first order approximation of spectral graph convolutions, yielding the following layer architecture (detailed in appendix A):

$$\mathbf{H}^{(t+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(t)} \mathbf{W}^{(t)}\right), \tag{1}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ has added self-connections, $\tilde{D}_{ii} = \sum_k \tilde{A}_{ik}$ defines its diagonal degree matrix, σ is a non-linearity such as sigmoid, $\tanh$ or $\mathrm{ReLU} = \max(0, \cdot)$, and $\mathbf{W}^{(t)}$ and $\mathbf{H}^{(t)}$ are the parameter and activation matrices of layer t respectively, with $\mathbf{H}^{(0)} = \mathbf{X}$ the input feature matrix.

3.2 Tools for a Hyperbolic GCN

Learning a parametrized function f_θ that respects hyperbolic geometry has been studied in Ganea et al. (2018b). More precisely, the authors consider feed forward neural networks as well as recurrent neural networks and derive hyperbolic counterparts to essential building blocks such as dense layers and logits in hyperbolic space. We leverage their results in order to generalize Graph Neural Networks.

3.3 Hyperbolic Right Matrix Multiplication

The first needed ingredient is a definition of **right matrix multiplication** $\mathbf{XW}$, where $\mathbf{X} \in \mathbb{R}^{n\times d}$ denotes the embedding and $\mathbf{W} \in \mathbb{R}^{d\times e}$ are the weights. We assume that each row $\mathbf{X}_{i\bullet} \in \mathbb{P}^d$ holds the hyperbolic embedding of node i and $\mathbf{W}$ is an Euclidean weight matrix. Let us first understand what a right matrix multiplication is doing in Euclidean space. The Euclidean right multiplication can be written column-wise $(\mathbf{XW})_{\bullet i}$ as follows:

$$(\mathbf{XW})_{\bullet i} = \begin{bmatrix} \mathbf{X}_{1\bullet}^T \mathbf{W}_{\bullet i} \\ \vdots \\ \mathbf{X}_{n\bullet}^T \mathbf{W}_{\bullet i} \end{bmatrix} = \begin{bmatrix} X_{11}W_{1i} + \cdots + X_{1d}W_{di} \\ \vdots \\ X_{n1}W_{1i} + \cdots + X_{nd}W_{di} \end{bmatrix} = W_{1i}\mathbf{X}_{\bullet 1} + \cdots + W_{di}\mathbf{X}_{\bullet d}$$

This leads to a new feature $(\mathbf{XW})_{\bullet i}$ by linear combining the features of $\mathbf{X}$ using the weights $W_{1i}, \dots, W_{di}$. The features viewed as vectors $X_{\bullet j}$ are not elements of hyperbolic space but rather elements of $\mathbb{R}^d$. It hence makes sense to lift this operation to the tangent space $\mathcal{T}_0\mathbb{P}^d \cong \mathbb{R}^d$, as suggested by Ganea et al. (2018b):

Definition 3.1. *Given an embedding $\mathbf{X} \in \mathbb{R}^{n\times d}$ holding hyperbolic embeddings in its rows and weights $\mathbf{W} \in \mathbb{R}^{d\times e}$, the **hyperbolic right matrix multiplication** is defined as*

$$(\mathbf{X}\otimes^c_{\mathbb{M}}\mathbf{W})_{i\bullet} = \exp_0^c\left((\log_0^c(\mathbf{X})\mathbf{W})_{i\bullet}\right) = \frac{1}{\sqrt{c}}\tanh\left(\frac{||(\mathbf{XW})_{i\bullet}||}{||\mathbf{X}_{i\bullet}||}\tanh^{-1}(\sqrt{c}||\mathbf{X}_{\bullet}i||)\right)\frac{(\mathbf{XW})_{i\bullet}}{||(\mathbf{XW})_{i\bullet}||}$$

where $\exp_0^c$ and $\log_0^c$ denote the exponential and logarithmic map in the Poincaré model.

This definition is in perfect agreement with the hyperbolic scalar multiplication, which can also be written as $r\otimes_{\mathbb{M}}x = \exp_0^c(r\log_0^c(x))$. This multiplication has the following desirable properties Ganea et al. (2018b). Another important element of any feed forward neural network is the application of a **non-linearity**. This is solved in a similar fashion in Ganea et al. (2018b). Given any map $\sigma : \mathbb{R}^d \to \mathbb{R}^d,\ x \mapsto \sigma(x)$, we define its hyperbolic dual as $\sigma^{\mathbb{M}} : \mathbb{P}^d \to \mathbb{P}^d,\ x \mapsto \exp_0^c(\sigma(\log_0^c(x)))$.

3.4 Hyperbolic Left Matrix Multiplication, Linear Combinations and Midpoints

For graph neural networks we also need the notion of message passing among neighboring nodes. We hence need an operation that combines different node embeddings and is permutation invariant. In Euclidean space this operation was achieved by the left multiplication of the embedding with the preprocessed adjacency $\hat{\mathbf{A}}$: $\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{Z}^{(l)}\right)$ where $\mathbf{Z}^{(l)} = \mathbf{H}^{(l)}\mathbf{W}^{(l)}$ Let us consider the left multiplication in more detail. For ease of notation we denote the preprocessed adjacency as $\mathbf{A} \in \mathbb{R}^{n\times n}$. The matrix product is then

$$(\mathbf{AX})_{i\bullet} = A_{i1}\mathbf{X}_{1\bullet} + \cdots + A_{in}\mathbf{X}_{n\bullet}$$

This means that the new representation of node i is obtained by calculating the linear combination of all the other node embeddings, weighted by the i-th row of $\mathbf{A}$. Notice that we have already seen this when studying the architecture of GCN on the node level.
To adapt GCNs to hyperbolic space we hence need a notion of taking a **weighted linear combination** or **barycentric coordinates**. Obviously, the Euclidean operation will not suffice as it is not respecting the underlying geometry nor restricted to $\mathbb{P}^d_c$.

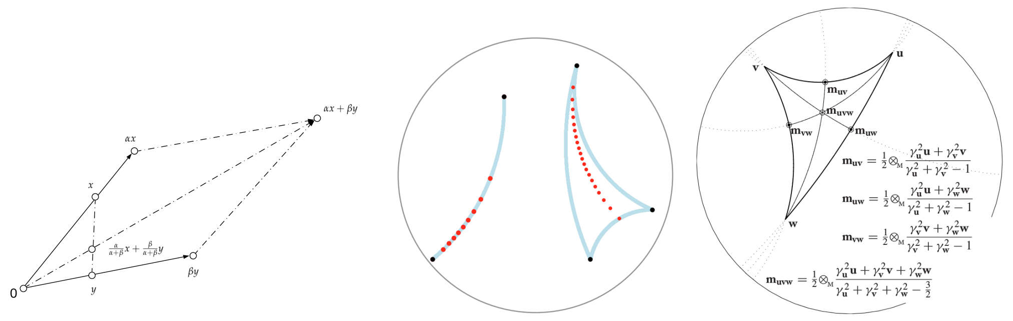
$\alpha x + \beta y$
αx
x
$\frac{\alpha}{\alpha+\beta}x + \frac{\beta}{\alpha+\beta}y$
βy
y
0
$\mathbf{u}$
$\mathbf{v}$
$\mathbf{w}$
$\mathbf{m_{uv}}$
$\mathbf{m_{uvw}}$
$\mathbf{m_{vw}}$
$\mathbf{m_{uw}}$
$\mathbf{m_{uv}} = \frac{1}{2}\otimes_{\mathbb{M}} \frac{\gamma_{\mathbf{u}}^2\mathbf{u} + \gamma_{\mathbf{v}}^2\mathbf{v}}{\gamma_{\mathbf{u}}^2 + \gamma_{\mathbf{v}}^2 - 1}$
$\mathbf{m_{uw}} = \frac{1}{2}\otimes_{\mathbb{M}} \frac{\gamma_{\mathbf{u}}^2\mathbf{u} + \gamma_{\mathbf{w}}^2\mathbf{w}}{\gamma_{\mathbf{u}}^2 + \gamma_{\mathbf{w}}^2 - 1}$
$\mathbf{m_{vw}} = \frac{1}{2}\otimes_{\mathbb{M}} \frac{\gamma_{\mathbf{v}}^2\mathbf{v} + \gamma_{\mathbf{w}}^2\mathbf{w}}{\gamma_{\mathbf{v}}^2 + \gamma_{\mathbf{w}}^2 - 1}$
$\mathbf{m_{uvw}} = \frac{1}{2}\otimes_{\mathbb{M}} \frac{\gamma_{\mathbf{u}}^2\mathbf{u} + \gamma_{\mathbf{v}}^2\mathbf{v} + \gamma_{\mathbf{w}}^2\mathbf{w}}{\gamma_{\mathbf{u}}^2 + \gamma_{\mathbf{v}}^2 + \gamma_{\mathbf{w}}^2 - \frac{3}{2}}$

Figure 3: Left: Euclidean Linear combination $\alpha x + \beta y$. Middle: Poincaré gyromidpoints (red dots) of two points for different weights on the left and Poincaré gyromidpoints in a hyperbolic triangle on the right with two equal and one free weight. Right: Möbius gyromidpoint in the Poincaré model defined by Ungar (2008) and alternatively, here in definition 3.2.

In order to introduce a hyperbolic linear combination, let us first look at the Euclidean version for only two vectors $x, y \in \mathbb{R}^d$ and weights $\alpha, \beta \in \mathbb{R}$: $z = \alpha x + \beta y$ This operation is visualized in fig. 3:

The resulting sum $\alpha x + \beta y$ is actually the weighted midpoint $m_{\mathbb{E}}(x, y; \alpha, \beta) = \frac{\alpha}{\alpha+\beta}x + \frac{\beta}{\alpha+\beta}y$ scaled by the factor $\alpha + \beta$. Scaling a vector in Euclidean space has the effect that the length of the vector is scaled by the same factor but the direction remains unchanged.
We can achieve the same scaling behaviour in hyperbolic space using the hyperbolic scalar multiplication $\otimes_{\mathbb{M}}^c$ since it has the similar property $||r \otimes_{\mathbb{M}}^c x||_{\mathbb{M}} = d_c^{\mathbb{M}}(0, r \otimes_{\mathbb{M}}^c x) = r||x||_{\mathbb{M}}$. Since geodesics through 0 are straight lines in the Poincaré model we also are not changing the direction. We hence reduced the problem to finding a notion of a weighted midpoint in hyperbolic space.

We will use the notion of a weighted midpoint arising from gyro theory, introduced in Ungar (2010):
Definition 3.2. *For $\{x_1, \dots, x_n\} \subset \mathbb{P}_c^d$ and weights $\{\alpha_1, \dots, \alpha_n\} \subset \mathbb{R}$ the **weighted gyromidpoint** in the Poincaré model is given by*

$$m_{\mathbb{M}}^c(x_1, \dots, x_n; \alpha_1, \dots \alpha_n) = \frac{1}{2} \otimes_{\mathbb{M}}^c \left(\sum_{i=1}^n \frac{\alpha_i \lambda_{x_i}^c}{\sum_{j=1}^n \alpha_j (\lambda_{x_j}^c - 1)} x_i \right)$$

Note that this definition of weighted midpoint implicitly performs a normalization by the term $\sum_{i=1}^n \alpha_i$ and we will also assume this for the Euclidean weighted mean. As can be seen in fig. 3, the gyromidpoints respect hyperbolic geometry as they follow the geodesics. This midpoint also has some analogous properties to the Euclidean version, as seen in Ungar (2010):
Lemma 1. *For $\{x_1, \dots, x_n\} \subset \mathbb{P}_c^d$ and $y \in \mathbb{P}_c^d$ and weights $\{\alpha_1 \dots, \alpha_n\} \subset \mathbb{R}$ it holds:*

- $m_{\mathbb{M}}^c(y \oplus_{\mathbb{M}}^c x_1, \dots, y \oplus_{\mathbb{M}}^c x_n; \alpha_1, \dots \alpha_n) = y \oplus_{\mathbb{M}}^c m_{\mathbb{M}}^c(x_1, \dots, x_n; \alpha_1, \dots \alpha_n)$
- $d_{\mathbb{M}}^c(x_1, m_{\mathbb{M}}^c(x_1, x_2; \frac{1}{2}, \frac{1}{2})) = \frac{1}{2} d_{\mathbb{M}}^c(x_1, x_2)$
- $m_{\mathbb{M}}^c(x_1, \dots, x_n; \alpha_1, \dots \alpha_n) \xrightarrow{c \longrightarrow 0} m_{\mathbb{E}}(x_1, \dots, x_n; \alpha_1, \dots \alpha_n)$

We are now ready to define the hyperbolic linear combination in the Poincaré model.
Definition 3.3. *For $\{x_1, \dots, x_n\} \subset \mathbb{P}_c^d$ and weights $\{\alpha_1, \dots, \alpha_n\} \subset \mathbb{R}$, the **hyperbolic linear combination** in the Poincaré model is given by: $(\sum_{i=1}^n \alpha_i) \otimes_{\mathbb{M}}^c m_{\mathbb{M}}^c(x_1, \dots, x_n; \alpha_1, \dots \alpha_n)$.*

We can further introduce the left matrix multiplication in the Poincaré model:
Definition 3.4. *Given $\mathbf{X} \in \mathbb{R}^{n \times d}$ with rows $\mathbf{X}_{i\bullet} \in \mathbb{P}_c^d$ and $\mathbf{A} \in \mathbb{R}^{n \times n}$, the **hyperbolic left matrix multiplication** $\mathbf{Z} = \mathbf{A} \otimes_{\mathbb{M}}^c \mathbf{X}$ is defined row-wise via*

$$\mathbf{Z}_{i\bullet} = (\sum_{j=1}^n A_{ij}) \otimes_{\mathbb{M}}^c m_{\mathbb{M}}^c(\mathbf{X}_{1\bullet}, \dots, \mathbf{X}_{n\bullet}; A_{i1}, \dots A_{in})$$

Lemma 2. *Given $\mathbf{X} \in \mathbb{R}^{n\times d}$ with rows $\mathbf{X}_{i\bullet} \in \mathbb{P}_c^d$, $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $r \in \mathbb{R}$, the hyperbolic left matrix multiplication satisfies: i) $\mathbf{I} \otimes_{\mathbb{M}}^c \mathbf{X} = \mathbf{X}$; ii) $r \otimes_{\mathbb{M}}^c (\mathbf{A} \otimes_{\mathbb{M}}^c \mathbf{X}) = (r\mathbf{A}) \otimes_{\mathbb{M}}^c \mathbf{X}$; iii) $\mathbf{A} \otimes_{\mathbb{M}}^c \mathbf{X} \xrightarrow{c\to 0} \mathbf{AX}$.*

3.5 Hyperbolic Logits

Finally, we need the logit and softmax layer, a neccessity for any classification task. We here use the model of Ganea et al. (2018b). More details are given in appendix E.

3.6 Hyperbolic GCN

We are now ready to introduce a hyperbolic version of GCN Kipf & Welling (2017), denoted by $\mathbb{H}_c$-GCN. Assume we are given a graph with node level features $G = (V, \mathbf{A}, \mathbf{X})$ where $\mathbf{X} \in \mathbb{R}^{n\times d}$ with each row $\mathbf{X}_{i\bullet} \in \mathbb{P}_c^d$ and adjacency $\mathbf{A} \in \mathbb{R}^{n\times n}$. We first perform a preprocessing step by mapping the Euclidean features to the Poincaré ball via the projection $\mathbf{X} \mapsto \frac{\mathbf{X}}{2\sqrt{c}||\mathbf{X}||_{max}}$, where $||\mathbf{X}||_{max}$ denotes the maximal norm among all hyperbolic embeddings in $\mathbf{X}$. For $l \in \{0, \dots, L-2\}$, the $(l+1)$-th layer of $\mathbb{H}_c - GCN$ is given by

$$\mathbf{H}^{(l+1)} = \sigma^{\mathbb{M}}\left(\hat{\mathbf{A}} \otimes_{\mathbb{M}}^c \left(\mathbf{H}^{(l)} \otimes_{\mathbb{M}}^c \mathbf{W}^{(l)}\right)\right)$$

where $\mathbf{H}^{(0)} = \mathbf{X}$, σ is some non-linearity and $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$. The final layer is a hyperbolic logit layer:

$$\mathbf{H}^{(L)} = \text{softmax}\left(\hat{\mathbf{A}}\ \text{logit}\left(\mathbf{H}^{(L-1)}, \mathbf{W}^{(L-1)}\right)\right)$$

A very important property of $\mathbb{H}_c - GCN$ is that its architecture recovers the Euclidean GCN when we let curvature go to zero: $\mathbb{H}_c$-GCN $\xrightarrow{c\to 0}$ GCN.

3.7 GCNs in Product of Constant Curvature Spaces

The extension of the above approach to spherical spaces and products of spaces of constant curvature is rather straightforward. For clarity, we detail it in appendix F.

4 Experiments

We evaluate the architectures introduced in the previous sections on the tasks of node classification and minimizing embedding distortion for several synthetic as well as real datasets.

More details are give in appendix G.

Minimizing Distortion Our first goal is to evaluate the graph embeddings learned by our GCN models on the representation task of fitting the graph metric in the embedding space. We desire to minimize the average distortion defined similarly as in Gu et al. (2019): $\frac{1}{n^2}\sum_{i,j}\left(\left(\frac{d(x_i,x_j)}{d_G(i,j)}\right)^2 - 1\right)^2$, where $d(x_i, x_j)$ is the distance between the embeddings of nodes i and j, while $d_G(i, j)$ is their graph distance (shortest path length).

We create three synthetic datasets as follows. Tree: a balanced tree of depth 5 and branching factor 4 consisting of 1365 nodes and 2728 edges. Torus: We sample points (nodes) from the (planar) torus, i.e. from the unit connected square; two nodes are connected by an edge iff their toroidal distance (the warped distance) is smaller than a fixed R = 0.01; this gives 1000 nodes and 30626 edges. Spherical Graph: we sample points (nodes) from $\mathbb{S}^2$, connecting nodes iff their distance is smaller than 0.2, leading to 1000 nodes and 17640 edges.

For the GCN models, we use 1-hot initial node features. We use two GCN layers with dimensions 16 and 10. The non-Euclidean models do not use additional non-linearities between layers. All euclidean parameters are updated using the ADAM optimizer with learning rate 0.01. Curvatures are learned using (stochastic) gradient descent and learning rate of 0.0001. All models are trained for 10000 epochs and we report the minimal achieved distortion. The results shown in table 1 reveal the benefit of our models. One can notice that estimated curvatures correspond to our geometric knowledge about these specific datasets.

Model	Tree	Toroidal Graph	Spherical Graph
GCN (Linear)	0.045	0.0607	0.0415
GCN (ReLU)	0.0502	0.0603	0.0409
$\mathbb{H}^{10}$-GCN	**0.0029**	0.272	0.267
$\mathbb{S}^{10}$-GCN	0.473	0.0485	**0.0337**
$\mathbb{H}^5 \times \mathbb{H}^5$-GCN	0.0048	0.112	0.152
$\mathbb{S}^5 \times \mathbb{S}^5$-GCN	0.51	**0.0464**	0.0359
$\left(\mathbb{H}^2\right)^4$- GCN	0.025	0.084	0.062
$\left(\mathbb{S}^2\right)^4$- GCN	0.312	0.0481	0.0378

Table 1: Minimum achieved average distortion of the different models. $\mathbb{H}$ and $\mathbb{S}$ denote hyperbolic and spherical models respectively.

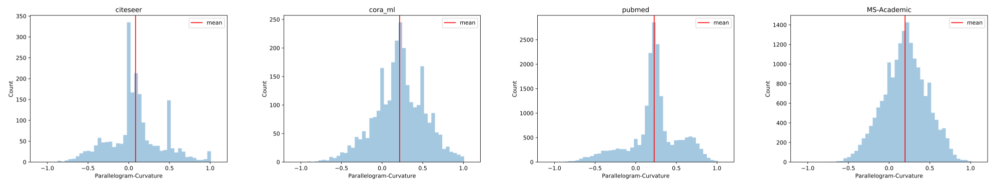

Figure 4: Histogram of Curvatures from "Deviation of Parallogram Law"

4.1 Node Classification

We consider the popular node classification datasets Citeseer Sen et al. (2008), Cora-ML McCallum et al. (2000) and Pubmed Namata et al. (2012) which are citation networks. Node labels correspond to the particular subfield the published document is associated with. Dataset statistics and training details are deffered to the appendix G due to the lack of space.

Curvature Estimations of Datasets To understand how far are the real graphs of the above datasets from the Euclidean geometry, we first estimate the graph curvature of the four studied datasets using the **deviation from the Parallelogram Law** Gu et al. (2019). This is detailed in appendix H. Curvature histograms are shown in fig. 4. It can be noticed that the datasets are mostly non-Euclidean. Thus, we have a good motivation to apply our constant-curvature GCN architectures.

Node classification results. These are shown in table 2. It can be seen that our models sometimes outpeform the two Euclidean GCN considered (with or without non-linearities), showcasing the benefit of our proposed architecture.

5 Conclusion

In this paper, we introduced natural extensions of graph neural networks to the non-Euclidean setting. To this end, we studied product of spaces of constant sectional curvature and equipped them with

Model	Citeseer	Cora-ML	Pubmed	MS Academic
GCN (ReLU)	75.7 ± 0.36	83.31 ± 0.36	**79.05 ± 0.52**	92.14 ± 0.25
GCN (Linear)	**76.28 ± 0.30**	83.81 ± 0.35	78.94 ± 0.50	**92.3 ± 0.21**
$\mathbb{H}_1^{64}$-GCN	**76.29 ± 0.3**	83.6 ± 0.34	79.01 ± 0.58	92.06±0.22
$\mathbb{S}_1^{64}$-GCN	76.18 ± 0.37	**83.97 ± 0.31**	**79.04 ± 0.5**	92.1±0.31
Prod-GCN	75.91 ± 0.34	82.9 ± 0.6	78.7 ± 0.53	91.9 ± 0.40

Table 2: Node classification: Average accuracy across 10 splits with estimated uncertainties at 95 percent confidence level via bootstrapping on our datasplits. $\mathbb{H}$ and $\mathbb{S}$ denote hyperbolic (Poincaré ball model) and spherical (stereographic projection) models respectively.

additional structure in the form of gyrovector spaces and matrix multiplications. Empirically we showed the benefit of our model on several real and synthetic datasets.

REFERENCES

Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. *N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification*. International Workshop on Mining and Learning with Graphs (MLG), 2018.

Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191. American Mathematical Soc., 2007.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Jie Chen, Tengfei Ma, and Cao Xiao. *Fastgcn: fast learning with graph convolutional networks via importance sampling*. ICLR, 2018.

Hyunghoon Cho, Benjamin DeMeo, Jian Peng, and Bonnie Berger. Large-margin classification in hyperbolic space. *AISTATS*, 2019.

Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. *Hyperspherical Variational Auto-Encoders*. Uncertainty in Artificial Intelligence (UAI), 856- 865, 2018.

Christopher De Sa, Albert Gu, Christopher Ré, and Frederic Sala. Representation tradeoffs for hyperbolic embeddings. 2018. URL `https://www.cs.cornell.edu/~cdesa/papers/arxiv2018_hyperbolic.pdf`.

Michaël Defferrard, Xavier Bresson, and Pierre Vand ergheynst. *Convolutional neural networks on graphs with fast localized spectral filtering*. In Advances in neural information processing systems (NIPS), 2016.

Michel Deza and Monique Laurent. *Geometry of Cuts and Metrics*. Springer, Vol. 15, 1996.

F. Ficken. *The Riemannian and affine differential geometry of product-spaces*. Annals of Mathematics, pp. 892–913, 1939.

Octavian Ganea, Gary Bécigneul, and Thomas Hofmann. *Hyperbolic entailment cones for learning hierarchical embeddings*. Proceedings of the thirty-fifth international conference on machine learning (ICML), 2018a.

Octavian Ganea, Gary Bécigneul, and Thomas Hofmann. *Hyperbolic neural networks*. In Advances in Neural Information Processing Systems, 2018b.

Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic entailment cones for learning hierarchical embeddings. In *International Conference on Machine Learning (ICML)*, 2018a.

Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2018b.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. *Neural Message Passing for Quantum Chemistry*. Proceedings of the International Conference on Machine Learning, 2017.

Daniele Grattarola, Daniele Zambon, Cesare Alippi, and Lorenzo Livi. Learning graph embeddings on constant-curvature manifolds for change detection in graph streams. *stat*, 1050:16, 2018.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649. IEEE, 2013.

Mikhael Gromov. Hyperbolic groups. In *Essays in group theory*, pp. 75–263. Springer, 1987.

Albert Gu, Frederic Sala, Beliz Gunel, and Christopher Ré. Learning mixed-curvature representations in product spaces. 2019.

Albert Gu, Frederic Sala, Beliz Gunel, and Christopher Ré. *Learning mixed-curvature representations in products of model spaces*. ICLR, 2019.

Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, et al. Hyperbolic attention networks. *arXiv preprint arXiv:1805.09786*, 2018.

Matthias Hamann. On the tree-likeness of hyperbolic spaces. *Mathematical Proceedings of the Cambridge Philosophical Society*, pp. 1–17, 2017. doi: 10.1017/S0305004117000238.

Matthias Hamann. *On the tree-likeness of hyperbolic spaces*. Mathematical Proceedings of the Cambridge Philo- sophical Society, pp. 117, 2017.

William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. In Advances in Neural Information Processing Systems, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pp. 173–182. International World Wide Web Conferences Steering Committee, 2017.

Christopher Hopper and Ben Andrews. *The ricci flow in riemannian geometry*. 2010.

Svante Janson. *Riemannian geometry: some examples, including map projections* . 2015.

Diederik P. Kingma and Jimmy Ba. *ADAM: A method for stochastic optimization*. ICLR, 2015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. ICLR, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. *Predict then propagate: graph neural networks meet personalized pagerank*. ICLR, 2019.

Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguná. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010.

John Lamping, Ramana Rao, and Peter Pirolli. A focus+ context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 401–408. ACM Press/Addison-Wesley Publishing Co., 1995.

Emile Mathieu, Charline Le Lan, Chris J Maddison, Ryota Tomioka, and Yee Whye Teh. Hierarchical representations with poincar\'e variational auto-encoders. *arXiv preprint arXiv:1901.06033*, 2019.

Jiri Matousek. *Lecture notes on metric embeddings*. 2013.

Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. *Automating the construction of internet portals with machine learning*. Information Retrieval, 3(2):127–163, 2000.

Galileo Namata, Ben London, Lise Getoor, and Bert Huang. *Query-driven Active Surveying for Collective Classification*. International Workshop on Mining and Learning with Graphs (MLG), 2012.

Maximilian Nickel and Douwe Kiela. *Poincare embeddings for learning hierarchical representations*. Advances in Neural Information Processing Systems, pp. 6341–6350, 2017.

Maximilian Nickel and Douwe Kiela. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In *International Conference on Machine Learning*, 2018.

Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems*, pp. 6341–6350, 2017.

Ivan Ovinnikov. Poincar\'e wasserstein autoencoder. *arXiv preprint arXiv:1901.01427*, 2019.

Joel Robbin and Dietmar Salamon. *Introduction to differential geometry*. ETH, Lecture Notes, 2011.

Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

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

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lisa Getoor, Brian Gallagher, and T. Eliassi-Rad. *Collective Classification in Network Data*. AI Magazine, 29(3):93–106, 2008.

Michael Spivak. *Comprehensive introduction to differential geometry*. volume four, 1979.

Michael Spivak. A comprehensive introduction to differential geometry. volume four. 1979.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

Alexandru Tifrea, Gary Bécigneul, and Octavian-Eugen Ganea. Poincaré glove: Hyperbolic word embeddings. 2019.

Pavan Turaga and Anuj Srivastava. *Riemannian computing in computer vision*. Springer, 2016.

Abraham Ungar. *A gyrovector space approach to hyperbolic geometry.* Synthesis Lectures on Mathematics and Statistics, 1(1):1–194, 2008.

Abraham Ungar. *Barycentric Calculus in Euclidean and Hyperbolic Geometry*. World Scientific, ISBN 9789814304931, 2010.

Abraham A Ungar. The hyperbolic pythagorean theorem in the poincaré disc model of hyperbolic geometry. *The American mathematical monthly*, 106(8):759–763, 1999.

Abraham A Ungar. *Analytic hyperbolic geometry: Mathematical foundations and applications*. World Scientific, 2005.

Abraham Albert Ungar. A gyrovector space approach to hyperbolic geometry. *Synthesis Lectures on Mathematics and Statistics*, 1(1):1–194, 2008.

Abraham Albert Ungar. Novel tools to determine hyperbolic triangle centers. In *Essays in Mathematics and its Applications*, pp. 563–663. Springer, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua. Bengio. *Graph Attention Networks.* ICLR, 2018.

Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pp. 639–655. Springer, 2012.

Richard C Wilson, Edwin R Hancock, Elżbieta Pekalska, and Robert PW Duin. Spherical and hyperbolic embeddings of data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2255–2269, 2014.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

Jiacheng Xu and Greg Durrett. Spherical latent spaces for stable variational autoencoders. *arXiv preprint arXiv:1808.10805*, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. *How powerful are graph neural networks*. ICLR, 2019.

Muhan Zhang and Yixin Chen. *Link prediction based on graph neural networks*. In Advances in Neural Information Processing Systems, 2018.

A GCN - A Brief Survey

A.1 Convolutional Neural Networks on Graphs

One of the pioneering works on neural networks in non-Euclidean domains was done by Defferrard et al. (2016). Their idea was to extend convolutional neural networks for graphs using tools from **graph signal processing**.

Given a graph $G = (V, \mathbf{A})$, where $\mathbf{A}$ is the adjacency matrix and V is a set of nodes, we define a signal on the nodes of a graph to be a vector $\mathbf{x} \in \mathbb{R}^n$ where x_i is the value of the signal at node i. Consider the diagonalization of the symmetrized graph Laplacian $\tilde{\mathbf{L}} = \mathbf{U}\Lambda\mathbf{U}^T$, where $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$. The eigenbasis $\mathbf{U}$ allows to define the graph Fourier transform $\hat{\mathbf{x}} = \mathbf{U}^T\mathbf{x} \in \mathbb{R}^n$.
In order to define a convolution for graphs, we shift from the vertex domain to the **Fourier domain**:

$$\mathbf{x} \star_G \mathbf{y} = \mathbf{U}\left(\left(\mathbf{U}^T\mathbf{x}\right) \odot \left(\mathbf{U}^T\mathbf{y}\right)\right)$$

Note that $\hat{\mathbf{x}} = \mathbf{U}^T\mathbf{x}$ and $\hat{\mathbf{y}} = \mathbf{U}^T\mathbf{y}$ are the graph Fourier representations and we use the element-wise product $\odot$ since convolutions become products in the Fourier domain. The left multiplication with $\mathbf{U}$ maps the Fourier representation back to a vertex representation.
As a consequence, a signal $\mathbf{x}$ filtered by g_θ becomes $\mathbf{y} = \mathbf{U}g_\theta(\Lambda)\mathbf{U}^T x$ where $g_\theta = \text{diag}(\theta)$ with $\theta \in \mathbb{R}^n$ constitutes a filter with all parameters free to vary. In order to avoid the resulting complexity $\mathcal{O}(n)$, Defferrard et al. (2016) replace the non-parametric filter by a polynomial filter:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

where $\theta \in \mathbb{R}^K$ resulting in a complexity $\mathcal{O}(K)$. Filtering a signal is unfortunately still expensive since $\mathbf{y} = \mathbf{U}g_\theta(\Lambda)\mathbf{U}^T x$ requires the multiplication with the Fourier basis $\mathbf{U}$, thus resulting in complexity $\mathcal{O}(n^2)$. As a consequence, Defferrard et al. (2016) circumvent this problem by choosing the **Chebyshev polynomials** T_k as a polynomial basis, $g_\theta(\Lambda) = \sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})$ where $\tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - \mathbf{I}$. As a consequence, the filter operation becomes $\mathbf{y} = \sum_{k=0}^{K} \theta_k T_k(\hat{\mathbf{L}})\mathbf{x}$ where $\hat{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{max}} - \mathbf{I}$. This led to a K-**localized** filter since it depended on the K-th power of the Laplacian. The **recursive** nature of these polynomials allows for an efficient filtering of complexity $\mathcal{O}(K|E|)$, thus leading to an computationally appealing definition of convolution for graphs. The model can also be built in an analogous way to CNNs, by stacking multiple convolutional layers, each layer followed by a non-linearity.

A.2 Graph Convolutional Networks

Kipf & Welling (2017) extended the work of Defferrard et al. (2016) and inspired many follow-up architectures Chen et al. (2018); Hamilton et al. (2017); Abu-El-Haija et al. (2018); Wu et al. (2019). The core idea of Kipf & Welling (2017) is to limit each filter to 1-hop neighbours by setting $K = 1$, leading to a convolution that is linear in the Laplacian $\hat{\mathbf{L}}$:

$$g_\theta \star \mathbf{x} = \theta_0\mathbf{x} + \theta_1\hat{\mathbf{L}}\mathbf{x}$$

They further assume $\lambda_{max} \approx 2$, resulting in the expression

$$g_\theta \star \mathbf{x} = \theta_0\mathbf{x} - \theta_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x}$$

To additionally alleviate overfitting, Kipf & Welling (2017) constrain the parameters as $\theta_0 = -\theta_1 = \theta$, leading to the convolution formula

$$g_\theta \star \mathbf{x} = \theta(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{x}$$

Since $\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ has its eigenvalues in the range $[0, 2]$, they further employ a reparametrization trick to stop their model from suffering from numerical instabilities:

$$g_\theta \star \mathbf{x} = \theta\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{x}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{D}_{ii} = \sum_{j=1}^{n} \tilde{A}_{ij}$.

Rewriting the architecture for multiple features $\mathbf{X} \in \mathbb{R}^{n \times d_1}$ and parameters $\Theta \in \mathbb{R}^{d_1 \times d_2}$ instead of $\mathbf{x} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$, gives

$$\mathbf{Z} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X} \Theta \in \mathbb{R}^{n \times d_2}$$

The final model consists of multiple stacks of convolutions, interleaved by a non-linearity σ:

$$\mathbf{H}^{(k+1)} = \sigma \left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(k)} \Theta^{(k)} \right)$$

where $\mathbf{H}^{(0)} = \mathbf{X}$ and $\Theta \in \mathbb{R}^{n \times d_k}$.

The final output $\mathbf{H}^{(K)} \in \mathbb{R}^{n \times d_K}$ represents the embedding of each node i as $\mathbf{h}_i = \mathbf{H}_{i\bullet} \in \mathbb{R}^{d_K}$ and can be used to perform node classification:

$$\hat{\mathbf{Y}} = \text{softmax} \left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(K)} \mathbf{W} \right) \in \mathbb{R}^{n \times L}$$

where $\mathbf{W} \in \mathbb{R}^{d_K \times L}$, with L denoting the number of classes.

In order to illustrate how embeddings of neighbouring nodes interact, it is easier to view the architecture on the **node level**. Denote by $\mathcal{N}(i)$ the neighbours of node i. One can write the embedding of node i at layer $k+1$ as follows:

$$h_i^{(k+1)} = \sigma \left(\Theta^{(l)} \sum_{j \in \mathcal{N}_i \cup \{i\}} \frac{h_j^{(k)}}{\sqrt{|\mathcal{N}(j)||\mathcal{N}(i)|}} \right)$$

Notice that there is no dependence of the weight matrices $\Theta^{(l)}$ on the node i, in fact the same **parameters are shared** across all nodes.
In order to obtain the new embedding $\mathbf{h}_i^{(k+1)}$ of node i, we average over all embeddings of the neighbouring nodes. This **Message Passing** mechanism gives rise to a very broad class of graph neural networks Kipf & Welling (2017); Veličković et al. (2018); Hamilton et al. (2017); Gilmer et al. (2017); Chen et al. (2018); Klicpera et al. (2019); Abu-El-Haija et al. (2018).

To be more precise, GCN falls into the more general category of models of the form

$$\begin{aligned} z_i^{(k+1)} &= \text{AGGREGATE}^{(k)}(\{h_j^{(k)} : j \in \mathcal{N}(i)\}; W^{(k)}) \\ h_i^{(k+1)} &= \text{COMBINE}^{(k)}(h_i^{(k)}, z_i^{(k+1)}; V^{(k)}) \end{aligned}$$

Models of the above form are deemed **Message Passing Graph Neural Networks** and many choices for AGGREGATE and COMBINE have been suggested in the literature Kipf & Welling (2017); Hamilton et al. (2017); Chen et al. (2018).

B Manifolds and Riemannian Geometry

In this section, we give an overview over the mathematical tools from differential geometry leveraged in this paper. More precisely, the notion of a manifold and Riemannian geometry will be discussed with a special emphasis on spherical geometry, hyperbolic space and its various models. For a deeper treatment of differential geometry we refer to Robbin & Salamon (2011); Spivak (1979); Hopper & Andrews (2010).

B.1 Manifolds and their Subclasses

Informally stated, manifolds are spaces that locally look like Euclidean space. An intuitive example is given by the sphere $\mathbb{S}^2 = \{x \in \mathbb{R}^3 : ||x|| = 1\}$. For someone standing on the Earth, space seems flat and therefore Euclidean. This only holds locally; globally the Earth obeys a different geometry as one can notice by watching ships disappear towards the horizon.
This resemblance with Euclidean space allows for many translations of important concepts from calculus to more general notions in manifolds. This locality property can be formalized:

Definition B.1. *A **topological manifold** $\mathcal{M}$ of dimension n is a Hausdorff, connected, topological space which is locally homeomorphic to $\mathbb{R}^n$. In other words, for every point $p \in \mathcal{M}$, one can find a neighbourhood $V_p \subset \mathcal{M}$ containing p and a homeomorphism ϕ, such that $U = \phi(V_p) \subset \mathbb{R}^n$.*

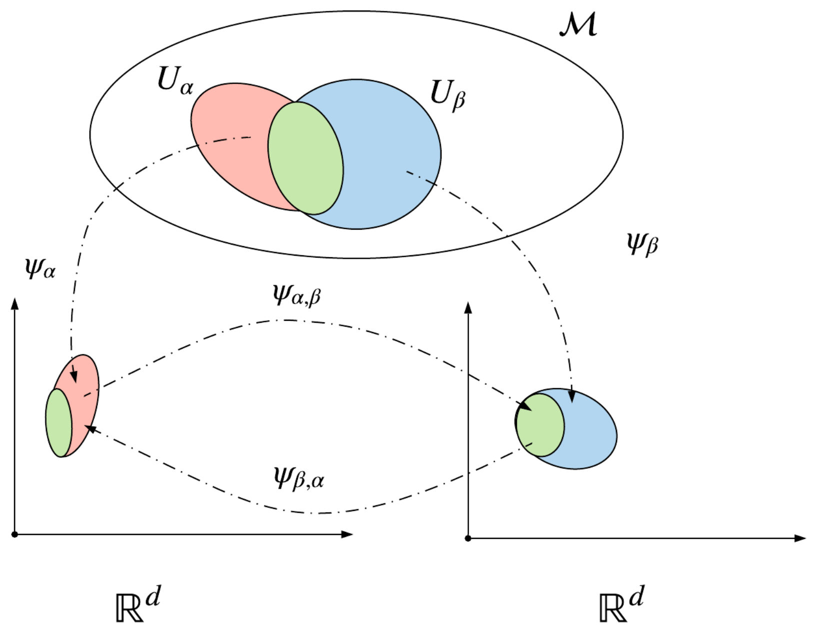

Figure 5: Two charts on $\mathcal{M}$ and their transition functions

There exist further subclasses of this definition of a manifold, each allowing for more additional structure. The class of manifolds that are mainly of interest in this work are **smooth manifolds**. In order to define this mathematical object, the notion of **charts** and **atlases** needs to be introduced.

Definition B.2. *A **chart** is a pair (V, ψ) where V is an open neighbourhood in $\mathcal{M}$ and $\psi : V \to U = \psi(V)$ is a homeomorphism onto $U \subset \mathbb{R}^n$.*

Definition B.3. *A collection of charts $\mathcal{A} = \{(U_i, \psi_i)\}_{i \in I}$ is called $\mathcal{C}^r$**-atlas** for $\mathcal{M}$ if:*

- $\mathcal{M} = \cup_{i \in I} U_i$
- *The transition functions $\psi_{ij} = \psi_i \circ \psi_j^{-1}$ restricted to the intersection $U_i \cap U_j$ are $\mathcal{C}^r$-differentiable.*

With these definitions at hand, the smooth manifold can be introduced:

Definition B.4. *A **smooth manifold** $\mathcal{M}$ is a topological manifold equipped with an equivalence class of atlases, whose transition functions are all smooth.*

These definitions are easier digested when accompanied with an **example**:

Example 1. *The **2-sphere** $\mathbb{S}^2 = \{(x, y, z) \in \mathbb{R}^3 : x^2 + y^2 + z^2 = 1\}$ is an example of a smooth manifold. One can see this by the following construction of an atlas:*

Define the open set

$$U_z^+ = \{(x, y, z) \in \mathbb{R}^3 : z > 0\}$$

and the open unit disk in $\mathbb{R}^2$:

$$\Omega_{x,y} = \{(x, y, z) \in \mathbb{R}^3 : x^2 + y^2 < 1\}$$

Further define the projection

$$p_z : U_z^+ \cap \mathbb{S}^2 \to \Omega_{x,y} \, , \, (x, y, z) \mapsto p_z(x, y, z) = (x, y)$$

with the inverse given by

$$p_z^{-1} : \Omega_{x,y} \to U_z^+ \cap \mathbb{S}^2 \, , \, (x, y) \mapsto p_z^{-1}(x, y) = (x, y, \sqrt{1 - x^2 - y^2})$$

Both functions are smooth, hence p_z constitutes a coordinate chart on $\mathbb{S}^2$. We can extend the same arguments to the other open sets $U_z^-, U_x^+, U_x^-, U_y^+, U_y^-$ with the appropriate disk and projection

chosen to obtain 5 *more charts. The union of these sets cover* $\mathbb{S}^2$ *and we thus have an atlas. We hence have proven that* $\mathbb{S}^2$ *is a topological manifold.*
Consider now the transition functions given by $f^{\pm}_{i,j} = p^{\pm}_i \circ (p^{\pm}_j)^{-1}$

For example

$$f^{+}_{x,y} = p^{+}_x \circ (p^{+}_y)^{-1} : U^{+}_x \cap U^{+}_y \mapsto \Omega_{x,z} \, , \, (x, y, z) \to (\sqrt{1 - x^2 - z^2}, z)$$

Again, one easily sees that this is a smooth map and one can further verify that this holds for any $f^{\pm}_{i,j}$. *As a result,* $\mathbb{S}^2$ *is a smooth manifold.*

B.2 Riemannian Manifolds

A very powerful subclass of smooth manifolds is given by **Riemannian manifolds**. The additional required structure is given by the so-called **Riemannian metric**. To understand this concept, we need to first visit the notion of tangent space.

B.2.1 Tangent Space

In differential geometry, there are usually two ways of looking at manifolds. The first way is viewing the manifold as an object embedded in an ambient space – extrinsic view. Again, the easiest example is the sphere $\mathbb{S}^2$ embedded in Euclidean space $\mathbb{R}^3$. The second view (intrinsic) is studying the manifold as an object itself, without relying on any ambient space. The second view is the preferred one in abstract differential geometry as it is mathematically more appealing to separate the manifold from its surrounding space and investigate it intrinsically. For machine learning, the first view is to be favored as one wants to use tools from Euclidean geometry which are present in the ambient space. As a result, there are two ways of introducing tangent spaces and for the aforementioned reasons. However, only the definition arising from the extrinsic point of view will be discussed here.

Definition B.5. *Consider* $\mathcal{M} \subset \mathbb{R}^n$ *a smooth manifold and a point* $p \in \mathcal{M}$. *A vector* $v \in \mathbb{R}^n$ *is called* ***tangential to*** p *if and only if there exists a smooth curve* $\gamma : \mathbb{R} \mapsto \mathcal{M}$ *fulfilling:*

- $\gamma(0) = p$
- $\dot{\gamma}(0) = v$

We denote by $\mathcal{T}_p\mathcal{M} = \{\dot{\gamma}(0) : \gamma : \mathbb{R} \mapsto \mathcal{M} \text{ is smooth, } \gamma(0) = p\}$ *the* ***tangent space*** *at* p.

This definition is extrinsic in the sense that it relies on the embedding space $\mathbb{R}^n$ by letting the tangent vectors "stick out" from the manifold (consider fig. 6 for instance). We will use the concept of tangent spaces to define the Riemannian metric.

B.2.2 Riemannian Metric

A **Riemannian metric** induces for every point $p \in \mathcal{M}$ an **inner product** in the tangent space $\mathcal{T}_p\mathcal{M}$, meaning a map $g_p : \mathcal{T}_p\mathcal{M} \times \mathcal{T}_p\mathcal{M} \to \mathbb{R}$ which is symmetric and positive definite:

- $g_p(x, y) = g_p(y, x)$ for every $x, y \in \mathcal{T}_p\mathcal{M}$
- $g_p(x, x) \geq 0$ for every $x \in \mathcal{T}_p\mathcal{M}$

Moreover, the collection of inner products varies **smoothly** in the location of the tangent space p. The best known example of a Riemannian metric is given by Euclidean space $\mathbb{R}^n$. The tangent space at p is given by $\mathcal{T}_p\mathbb{R}^n \cong \mathbb{R}^n$ and is equipped with the standard inner product $g_p(x, y) = x^T y$. We will usually identify g_p with its matrix representation. The Riemannian metric allows for the extension of many mathematical tools to Riemannian manifolds that are needed in order to study machine learning in non-Euclidean spaces.

B.3 Mathematical tools

Given a Riemannian metric g on a manifold $\mathcal{M}$, one is able to define many quantities analogous to Euclidean space.

B.3.1 Distance function

Given a curve $\gamma : [0,1] \to \mathcal{M}$, we can use the metric g to calculate the **length** of it:

$$l(\gamma) = \int_0^1 \sqrt{g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t))}dt$$

Given now two points $x, y \in \mathcal{M}$, we can define the **distance** between the two by calculating the infimum of the length over all curves γ fulfilling $\gamma(0) = x$ and $\gamma(1) = y$:

$$d(x,y) = \inf_{\gamma} l(\gamma) = \inf_{\gamma} \int_0^1 \sqrt{g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t))}dt \tag{2}$$

Notice that in the example of $\mathcal{M} = \mathbb{R}^n$, this formula reduces to

$$d(x,y) = \inf_{\gamma} \int_0^1 ||\dot{\gamma}(t)||dt = ||y - x||$$

where the second equality can be shown by the formalism of Euler-Lagrange. The curve that achieves the infimum in (2.1) is called **geodesic**. It is the natural extension of straight lines in Euclidean space, as they are also the distance minimizing curves between two points.

B.3.2 Gradients in the Tangent Spaces of Riemannian Manifolds

In order to be able to perform optimization, we need to compute **gradients** of a cost function. Again, we have to move to tangent spaces to overcome this problem. Given parameters $z \in \mathcal{M}$, we want to calculate a gradient that indicates the best direction we should take in order to reduce the value of the cost function $\mathcal{L}$ and update the parameters z of our model correspondingly. One could naively calculate the Euclidean gradient and perform the update

$$z \leftarrow z - \eta \nabla_z^E \mathcal{L}$$

There are two flaws in this scheme:

- The Euclidean gradient does not take into account the geometry of the manifold. We need the **Riemannian gradient**, that applies a correction according to the metric tensor:

$$\nabla_z^R = g_z^{-1} \nabla_z^E$$

- **The notion of a step** is not well-defined on $\mathcal{M}$ since there is no guarantee that the result of the subtraction lies again in $\mathcal{M}$. We hence need a manifold-specific way of taking a step.

The correct way of taking a step on a manifold is given by the exponential map, introduced in the next section.

B.3.3 Exponential and Logarithmic Map

Consider a point $p \in \mathcal{M}$ and $v \in \mathcal{T}_p\mathcal{M}$. The **unit speed geodesic** starting from x in direction v is the unique geodesic $\gamma_{x,v}$ with $\gamma(0) = x$ and $\dot{\gamma}(0) = v$. The **exponential map** $\exp_p(v)$ allows to map from the tangent space $\mathcal{T}_p\mathcal{M}$ to the manifold $\mathcal{M}$ and is defined as follows:

$$\exp_p : \mathcal{T}_p\mathcal{M} \mapsto \mathcal{M}, \ v \to \exp_p(v) = \gamma_{p,v}(1)$$

Its inverse, $\log_p : \mathcal{M} \mapsto \mathcal{T}_p\mathcal{M}$ is called **logarithmic map**. Notice that the definition of the exponential map depends on the unit speed geodesic. Unfortunately, closed-formulas for the latter are rather seldom or computationally very expensive. As a result, one sometimes has to rely on **cheaper** approximations to the exponential map, called **retractions**.
Finally, observe that for $\mathcal{M} = \mathbb{R}^n$, it holds that $\exp_x(v) = x + v$ and $\log_x(y) = y - x$.
The combination of the two previous sections leads to **Riemannian optimization** Bonnabel (2013).

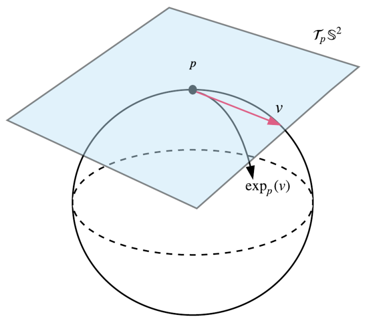

Figure 6: The exponential map on $\mathbb{S}^2$ applied to a tangential vector $v \in \mathcal{T}_p\mathbb{S}^2$

B.3.4 Riemannian Optimization

Denote by $\mathcal{L}$ a loss function and by $z \in \mathcal{M}$ parameters constrained to live on the manifold. The vanilla **gradient descent step** in Euclidean space is given by:

$$z \leftarrow z - \eta \nabla_z^E \mathcal{L}$$

In **Riemannian optimization** Bonnabel (2013), one replaces this update scheme by

$$z \leftarrow \exp_z(-\eta \nabla_z^{\mathcal{M}} \mathcal{L})$$

Note that for $\mathcal{M} = \mathbb{R}^n$, we recover again the vanilla gradient descent step. If calculating the exponential map is not possible or feasible, it is replaced by the aforementioned retraction R_z. For intuition, the Riemannian gradient $\nabla_z^{\mathcal{M}}$ indicates the best direction in the tangent space $\mathcal{T}_z\mathcal{M}$ and we map the resulting direction in the tangent space down to the manifold, ensuring $z \in \mathcal{M}$.

B.4 An Example: The Sphere $\mathbb{S}_R^{n-1}$

In order to provide further understanding and intuition for the introduced quantities, we will study the Riemannian manifold $\mathbb{S}_R^{n-1} = \{x \in \mathbb{R}^n : ||x|| = R\}$ equipped with the metric tensor $g_p = \mathbb{1}$ inherited from the ambient space $\mathbb{R}^n$. For $p \in \mathbb{S}_R^{n-1}$ the **tangent space** at p is given by

$$\mathcal{T}_p\mathbb{S}_R^{n-1} = \{x : x^T p = 0\}$$

This holds because for any smooth curve γ on $\mathbb{S}^{n-1}$, we have $||\gamma(t)||^2 = \gamma(t)^T\gamma(t) = R$, hence $\gamma(t)^T\dot{\gamma}(t) = 0$ and thus for $t = 0 : p^T v = 0$.

Next consider the **unit speed geodesics** on $\mathbb{S}_R^{n-1}$. Notice that although the metric tensor is the Euclidean one, the infimum in (2.1) is taken over less curves γ since we are restricted to the sphere. In particular, straight lines are not included in the set of allowed curves. The minimum is achieved by so-called **great circles**, parametrized by

$$\gamma_{x,v}(t) = \cos\left(\frac{||v||}{R}t\right)x + R\sin\left(\frac{||v||}{R}t\right)\frac{v}{||v||}$$

The induced **distance** on the sphere is given by

$$d_{\mathbb{S}_R^n}(x, y) = R\arccos\left(\frac{1}{R^2}x^T y\right)$$

which corresponds to the arc length of the great circle connecting the points $x, y \in \mathbb{S}^{n-1}$. Using the equation for the unit speed geodesic, one easily arrives at the formula for the **exponential map**:

$$\exp_x(v) = x\cos\left(\frac{1}{R}||v||\right) + R\sin\left(\frac{1}{R}||v||\right)\frac{v}{||v||}$$

The **logarithmic map** is given by:

$$\log_x(y) = \arccos(Rx^T y)\frac{y - \frac{1}{R^2}x^T yx}{||y - \frac{1}{R^2}x^T yx||}$$

One important observation is the following. Intuitively, one would expect that we recover Euclidean space again when the radius R of the sphere goes to infinity. This is due to the vanishing curvature $c = \frac{1}{R^2}$, since space starts to feel more flat the bigger the sphere gets. The problem is that the formulas for the distance, exponential map and all other quantities **do not recover** their Euclidean **counterpart**. This follows from the chosen parametrization, which lets the point on the manifold implicitly depend on the curvature c through the hard constraint $||x|| = \frac{1}{\sqrt{c}}$, making a recovery impossible. Convergence to Euclidean space will be important for the machine learning algorithms studied in this paper and as a result, we will work with a different isometric model of the sphere $\mathbb{S}^{n-1}$.

C Graph Embeddings in Non-Euclidean Geometries

In this section we will motivate non-Euclidean embeddings of graphs and show why the underlying geometry of the embedding space can be very beneficial for its representation. We first introduce a measure of how well a graph is represented by some embedding $f : V \to \mathcal{X},\ i \mapsto f(i)$:

Definition C.1. *Given an embedding $f : V \to \mathcal{X},\ i \mapsto f(i)$ of a graph $G = (V, A)$ in some metric space $\mathcal{X}$, we call f a **D-embedding** for $D \geq 1$ if there exists $r > 0$ such that*

$$r \cdot d_G(i, j) \leq d_{\mathcal{X}}(f(i), f(j)) \leq D \cdot r \cdot d_G(i, j)$$

*The infimum over all such D is called the **distortion** of f.*

The r in the definition of distortion allows for scaling of all distances. Note further that a perfect embedding is achieved when $D = 1$.

C.1 Trees and Hyperbolic Space

Trees are graphs that do not allow for a cycle, in other words there is no node $i \in V$ for which there exists a path starting from i and returning back to i without passing through any node twice. The number of nodes increases **exponentially** with the depth of the tree. This is a property that prohibits Euclidean space from representing a tree accurately. What intuitively happens is that "we run out of space". Consider the trees depicted in fig. 7. Here the yellow nodes represent the roots of each tree. Notice how rapidly we struggle to find appropriate places for nodes in the embedding space because their number increases just too fast.

Moreover, graph distances get **extremely distorted** towards the leaves of the tree. Take for instance the green and the pink node. In graph distance they are very far apart as one has to travel up all the way to the root node and back to the border. In Euclidean space however, they are very closely embedded in a L_2-sense, hence introducing a big error in the embedding.

This problem can be very nicely illustrated by the following theorem:

Theorem 2. *Consider the tree $K_{1,3}$ (also called 3-star) consisting of a root node with three children. Then every embedding $\{x_1, \dots, x_4\}$ with $x_i \in \mathbb{R}^k$ achieves at least distortion $\frac{2}{\sqrt{3}}$ for any $k \in \mathbb{N}$.*

Proof. We will prove this statement by using a special case of the so called **Poincaré-type inequalities** Deza & Laurent (1996):

For any $b_1, \dots, b_k \in \mathbb{R}$ with $\sum_{i=1}^{k} b_i = 0$ and points $x_1, \dots, x_k \in \mathbb{R}^n$ it holds that

$$\sum_{i,j=1}^{k} b_i b_j ||x_i - x_j||^2 \leq 0$$

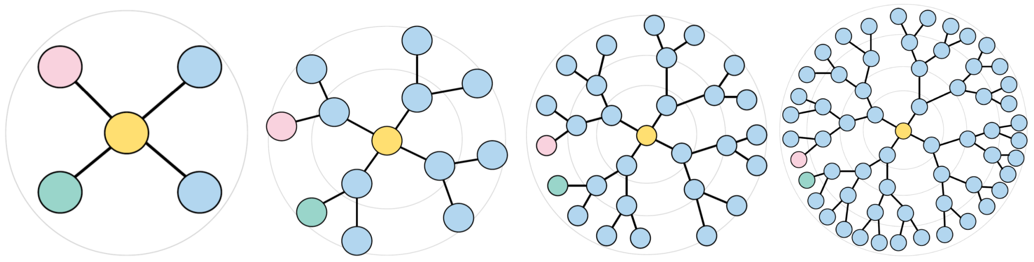

Figure 7: Euclidean embeddings of trees of different depths. All the four most inner circles are identical. Ideal node embeddings should match in distance the graph metric, e.g. the distance between the pink and green nodes should be the same as their shortest path length. Notice how we quickly run out of space, e.g. the pink and green nodes get closer as opposed to farther. This issue is resolved when embedding trees in hyperbolic spaces.

Consider now an embedding of the tree $x_1, \dots, x_4$ where x_1 represents the root node. Choosing $b_1 = -3$ and $b_i = 1$ for $i \neq 1$ leads to the inequality

$$||x_2 - x_3||^2 + ||x_2 - x_4||^2 + ||x_3 - x_4||^2 \leq 3||x_1 - x_2||^2 + 3||x_1 - x_3||^2 + 3||x_1 - x_4||^2$$

The left-hand side of this inequality in terms of the graph distance is

$$d_G(2,3)^2 + d_G(2,4)^2 + d_G(3,4)^2 = 2^2 + 2^2 + 2^2 = 12$$

and the right-hand side is

$$3 \cdot d_G(1,2)^2 + 3 \cdot d_G(1,3)^2 + 3 \cdot d_G(1,4)^2 = 3 + 3 + 3 = 9$$

As a result, we always have that the distortion is lower-bounded by $\sqrt{\frac{12}{9}} = \frac{2}{\sqrt{3}}$ □

Euclidean space thus already fails to capture the geometric structure of a very simple tree. This problem can be remedied by replacing the underlying Euclidean space by hyperbolic space.

Consider again the distance function in the Poincaré model, for simplicity with $c = 1$:

$$d_{\mathbb{P}}(x, y) = \cosh^{-1}\left(1 + 2\frac{||x - y||^2}{(1 - ||x||^2)(1 - ||y||^2)}\right)$$

Assume that the tree is embedded in the same way as in fig. 7, just restricted to lie in the disk of radius $\frac{1}{\sqrt{c}} = 1$. Notice that as soon as points move closer to the boundary ($||x|| \to 1$), the fraction explodes and the resulting distance goes to infinity. As a result, the further you move points to the border, the more their distance increases, exactly as nodes on different branches are more distant to each other the further down they are in the tree. We can express this advantage in geometry in terms of distortion:

Theorem 3. *There exists an embedding $x_1, \dots, x_4 \in \mathbb{P}^2$ for $K_{1,3}$ achieving distortion $1 + \epsilon$ for $\epsilon > 0$ arbitrary small.*

Proof. Since the Poincaré distance is invariant under Möbius translations we can again assume that $x_1 = 0$. Let us place the other nodes on a circle of radius r. Their distance to the root is now given as

$$d_{\mathbb{P}}(x_i, 0) = \cosh^{-1}\left(1 + 2\frac{||x_i||^2}{1 - ||x_i||^2}\right) = \cosh^{-1}\left(1 + 2\frac{r^2}{1 - r^2}\right)$$

By invariance of the distance under centered rotations we can assume w.l.o.g. $x_2 = (r, 0)$. We further embed

- $x_3 = \left(r\cos(\frac{2}{3}\pi), r\sin(\frac{2}{3}\pi)\right) = \left(-\frac{r}{2}, \frac{\sqrt{3}}{2}r\right)$
- $x_4 = \left(r\cos(\frac{4}{3}\pi), r\sin(\frac{4}{3}\pi)\right) = \left(-\frac{r}{2}, \frac{\sqrt{3}}{2}r\right)$.

This procedure gives:

$$d_{\mathbb{P}}(x_2, x_3) = \cosh^{-1}\left(1 + 2\frac{||\left(\frac{3r}{2}, \frac{-\sqrt{3}}{2}r\right)||^2}{(1-r^2)^2}\right) = \cosh^{-1}\left(1 + 2\frac{3r^2}{(1-r^2)^2}\right)$$

If we let the points now move to the border of the disk we observe that

$$\frac{\cosh^{-1}\left(1 + 2\frac{3r^2}{(1-r^2)^2}\right)}{\cosh^{-1}\left(1 + 2\frac{r^2}{1-r^2}\right)} \xrightarrow{r \longrightarrow 1} 2$$

But this means in turn that we can achieve distortion $1 + \epsilon$ for $\epsilon > 0$ arbitrary small. QED. □

The tree-likeliness of hyperbolic space has been investigated on a deeper mathematical level. Sarkar (2011) show that a similar statement as in Theorem 2 holds for a very general class of trees. The interested reader is referred to Hamann (2017); Sarkar (2011) for a more in-depth treatment of the subject.

Cycles are the subclasses of graphs that are not allowed in a tree. They consist of one path that reconnects the first and the last node: $(v_1, \dots, v_n, v_1)$. Again there is a very simple example of a cycle, hinting at the limits Euclidean space incurs when trying to preserve the geometry of these objects Matousek (2013).

Theorem 4. *Consider the cycle $G = (V, E)$ of length four. Then any embedding $(x_1, \dots, x_4)$ where $x_i \in \mathbb{R}^k$ achieves at least distortion $\sqrt{2}$.*

Proof. Denote by x_1, x_2, x_3, x_4 the embeddings in Euclidean space where x_1, x_3 and x_2, x_4 are the pairs without an edge. Again using the Poincaré-type inequality with $b_1 = b_3 = 1$ and $b_2 = b_4 = -1$ leads to the **short diagonal theorem** Matousek (2013):

$$||x_1 - x_3||^2 + ||x_2 - x_4||^2 \leq ||x_1 - x_2||^2 + ||x_2 - x_3||^2 + ||x_3 - x_4||^2 + ||x_4 - x_1||^2$$

The left hand side of this inequality in terms of the graph distance is $d_G(1,3)^2 + d_G(2,4)^2 = 2^2 + 2^2 = 8$ and the right hand side is $1^2 + 1^2 + 1^2 + 1^2 = 4$.
Therefore any embedding has to shorten one diagonal by at least a factor $\sqrt{2}$. □

It turns out that in spherical space, this problem can be solved perfectly in one dimension for any cycle.

Theorem 5. *Given a cycle $G = (V, E)$ of length n, there exists an embedding $\{x_1, \dots, x_n\}$ achieving distortion 1.*

Proof. We model the one dimension spherical space as the circle $\mathbb{S}^1$. Placing the points at angles $\frac{2\pi i}{n}$ and using the arclength on the circle as the distance measure leads to an embedding of distortion 1 as all pairwise distances are perfectly preserved. □

Notice that we could also use the exact same embedding in the two dimensional stereographic projection model with $c = 1$ and we would also obtain distortion 1. The difference to the Poincaré disk is that spherical space is finite and the border does not correspond to infinitely distant points. We therefore have no ϵ since we do not have to pass to a limit.

D GYROVECTOR SPACES

Euclidean space is so popular because it allows for a lot of structure. We have well-defined operations such as **addition** of vectors or **scalar multiplication** between a real number and a vector, which obey certain rules. Those operations are the building blocks for more sophisticated concepts like matrix multiplication, which in turn are the building blocks for many machine learning models. In order to have a principled extension of the algorithms of interest, we need a translation of these quantities to the non-Euclidean setting. Gyrovector spaces Ungar (2008) offer such a generalization and will be studied in the following section.

D.1 Vector Spaces and Gyrovector Spaces

We wish to extend the notion of Euclidean vector spaces.

For the Euclidean space $\mathbb{R}^d$, the addition and scalar multiplication are given by the well-known component-wise vector addition and scalar multiplication. The inner product is given by the usual dot product $x^T y$. It is clear that we have to give up on some part of the structure, as any real vector space of dimension d is isomorphic to $\mathbb{R}^d$. In order to answer how much we can retain, we first need to introduce a mathematical object residing on a deeper level, namely the equivalent to a group. This dual group will be the foundation for the extension of a vector space. It turns out that we already have to give up on some of the structure on the group level, as can be seen by the definition of a so called **gyrogroup** Ungar (2008):

Definition D.1. *A **gyrogroup** is a set G with well-defined binary operation $\oplus$ satisfying the following axioms:*

1. *There exists at least one $0 \in G$ such that for every $g \in G$: $0 \oplus g = g$*
2. *For every $g \in G$ there exists $\ominus g \in G$ such that $\ominus g \oplus g = 0$*
3. *For any $a, b, c \in G$ there exists a unique element $gyr[a, b]c \in G$ such that $a \oplus (b \oplus c) = (a \oplus b) \oplus gyr[a, b]c$*
4. *The map $c \rightarrow gyr[a, b]c$ is an automorphism satisfying $gyr[a, b] = gyr[a \oplus b, b]$*

We see that we lose the **associative law** and replace it by the weaker notion stated in 3 and 4. Equipped with this definition, we introduce the concept of a real gyrovector space Ungar (2008):

Definition D.2. *A real inner product **gyrovector space** is the triple $(G, \oplus, \otimes)$ where $(G, \oplus)$ is a gyrocommutative gyrogroup fullfilling the following axioms:*

1. *G is a subset of a real inner product space $\mathbb{V}$, inheriting the inner product $\langle x, y\rangle$ which is invariant under gyrations: $\langle gyr[x, y]a, gyr[x, y]b\rangle = \langle x, y\rangle$*
2. *There exists $1 \in G$ such that $1 \otimes x = x$ for any $x \in G$ (Identity)*
3. *$(s_1 + s_2) \otimes x = s_1 \otimes x \oplus s_2 \otimes x$ (Scalar Distributive Law)*
4. *$(s_1 s_2) \otimes x = s_1 \otimes (s_2 \otimes x)$ (Scalar Associativity)*
5. *$\frac{|s| \otimes x}{||s \otimes x||} = \frac{x}{||x||}$ (Scaling)*
6. *$gyr[x, y](s \otimes z) = s \otimes gyr[x, y]z$*
7. *$gyr[s_1 \otimes x, s_2 \otimes x] = id$*
8. *$||G|| = \{\pm||a|| : a \in G\}$ with $\oplus$ and $\otimes$ forms a vector space.*
9. *$||r \otimes x|| = |r| \otimes ||x||$ (Homogeneity)*
10. *$||x \oplus y|| \leq ||x|| \oplus ||y||$ (Gyrotriangle inequality)*

Note that we define $\oplus$ and $\otimes$ for the norms of $a \in G$ as seen in 9 and 10 in perfect analogy to the vectorized version, by treating the scalars as one dimensional vectors. In the following sections we will study two particular examples of manifolds that give rise to a gyrostructure, namely hyperbolic and spherical space. Together with Euclidean space, they are the only manifolds inducing a geometry with constant sectional curvature.

D.2 Hyperbolic Space and its Gyrostructure

There are several isometric models of hyperbolic geometry. Each model offers some advantages and disadvantages in its parametrization. It is important to notice that these advantages and disadvantages only arise through the chosen **parametrizations**, all the models being isometric.

In this work, we chose to work with the **Poincaré model**. Its conformality property and the **pioneering work** in this model by Nickel & Kiela (2017); Ganea et al. (2018a;b) facilitated the extension of the algorithms of consideration to the non-Euclidean setting.

D.2.1 Gyrovector Space in the Poincaré Ball

This section is based on the extensive research of Ungar (2008) on hyperbolic space and its underlying gyrovector structure, as well as Ganea et al. (2018b) who connected gyro theory and machine learning in their work. For proofs of the introduced results, we hence refer to the aforementioned sources.

Inspired by **general relativity** and **relativistic velocity addition**, Ungar (2008) discovered the notion of gyro theory. General relativity and hyperbolic space are very tightly related and it turns out that the relativistic velocity addition is exactly the gyro addition $\oplus_{\mathbb{E}}$ in the Klein model:

$$v \oplus_{\mathbb{E}}^{c} w = \frac{1}{1+cv^T w}\left(v(1+c\frac{\gamma_v}{1+\gamma_v}v^T w) + v\frac{1}{\gamma_v}\right)$$

This addition can be translated to the Poincaré ball, leading to the so called Möbius addition:

Definition D.3. *For $x, y \in \mathbb{P}_c^d$, the* ***Möbius addition*** *is given by:*

$$x \oplus_{\mathbb{M}}^{c} y = \frac{(1+2cx^Ty + c||y||^2)x + (1-c||x||^2)y}{1+2cx^Ty + c^2||x||^2||y||^2} \in \mathbb{P}_c^d$$

Notice that as desired, $x \oplus_{\mathbb{M}}^{c} y \xrightarrow{c \longrightarrow 0} x + y$ recovering Euclidean addition in the limit. We can further define a scalar multiplication:

Definition D.4. *For $s \in \mathbb{R}$ and $x \in \otimes_{\mathbb{M}}^{c}$, the* ***scalar multiplication*** *in the Poincaré model is given by:*

$$s \otimes_{\mathbb{M}}^{c} x = \frac{1}{\sqrt{c}} \tanh\left(s \cdot \tanh^{-1}(\sqrt{c}||x||)\right)\frac{x}{||x||} \in \mathbb{P}_c^d$$

Again observe that in the limit $s \otimes_{\mathbb{M}}^{c} x \xrightarrow{c \longrightarrow 0} sx$. One can show that Möbius addition and scalar multiplication in $\mathbb{P}_c^d$ form a gyrovector space Ungar (2008) as introduced in Definition D.2.

We will now list various results, illustrating how the role of Möbius addition and scalar multiplication mimics the one observed in Euclidean geometry with component-wise addition and multiplication.

We can use the Möbius addition to rewrite the distance formula:

Lemma 3. $d_{\mathbb{P}}^{c}(x, y) = \frac{2}{\sqrt{c}} \tanh^{-1}(\sqrt{c}|| - x \oplus_{\mathbb{M}}^{c} y||)$

Notice how the distance depends on x and y only through $|| - x \oplus_{\mathbb{M}}^{c} y||$, as is the case for Euclidean geometry where $\oplus_{\mathbb{M}}^{c}$ becomes $+$. This model is smoothly deformed to the Euclidean model in the curvature limit as $d_{\mathbb{P}}^{c}(x, y) \xrightarrow{c \longrightarrow 0} 2||x - y||$. Moreover, the introduced operations interact naturally with the underlying geometry, as can be seen by the following properties Ungar (2008):

Corollary 5.1. *For $x, y, z \in \mathbb{P}_c^d$ and $n \in \mathbb{N}$, it holds:*

- $d_{\mathbb{P}}^{c}(x \oplus_{\mathbb{M}}^{c} z, y \oplus_{\mathbb{M}}^{c} z) = d_{\mathbb{P}}^{c}(x, y)$
- $||r \otimes_{\mathbb{M}}^{c} x||_{\mathbb{P}} = r||x||_{\mathbb{P}}$ *where* $||x||_{\mathbb{P}} = d_{\mathbb{P}}^{c}(x, 0)$
- $n \otimes_{\mathbb{M}}^{c} x = x \oplus_{\mathbb{M}}^{c} \dots \oplus_{\mathbb{M}}^{c} x$, *where the addition is performed n times.*

One can see that Möbius translations leave the distance invariant and Möbius scaling on the other hand transforms the norm $||x||_{\mathbb{P}}$ appropriately, in perfect duality to the Euclidean world. We can also write the formula for **geodesics** in the Poincaré model in an intuitive manner Ungar (2008):

Lemma 4. *For $x, y \in \mathbb{P}_c^d$, the geodesic $\gamma_{x \longrightarrow y} : \mathbb{R} \to \mathbb{P}_c^d$, connecting x and y is given by*

$$\gamma_{x \longrightarrow y}(t) = x \oplus_{\mathbb{M}}^{c} \left(t \otimes_{\mathbb{M}}^{c} (-x \oplus_{\mathbb{M}}^{c} y)\right)$$

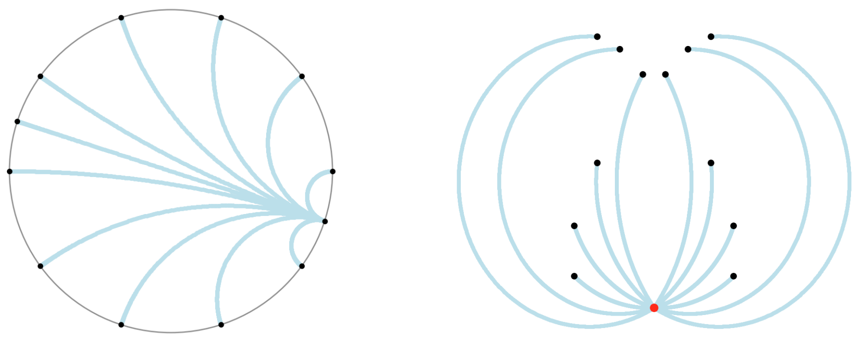

Figure 8: Geodesics in the Poincaré disk (left) and in the stereographic projection model of the sphere $\mathbb{S}^2$ (right).

This is in direct analogy to the Euclidean geodesics parametrized by $\gamma_{x\to y}(t) = x + t(y - x)$, where one simply replaces all Möbius operations by the Euclidean ones. As a consequence of this formula, we recover again the Euclidean geodesics for $c \to 0$.

Finally, we also obtain the **exponential** and **logarithmic map** in closed form Ganea et al. (2018b):

Lemma 5. *For $x \in \mathbb{P}_c^d$ and $v \neq 0$, the exponential map $\exp_x : T_x\mathbb{P}_c^d \to \mathbb{P}_c^d$ is given by*

$$\exp_x^c(v) = x \oplus_{\mathbb{M}}^c \left(\tanh\left(\sqrt{c}\frac{\lambda_x^c ||v||}{2}\right) \frac{v}{||v||} \right)$$

Moreover for $y \neq x$, $\log_x : \mathbb{P}_c^d \to T_x\mathbb{P}_c^d$ can be expressed via

$$\log_x^c(y) = \frac{2}{\sqrt{c}\lambda_x^c} \tanh^{-1}(\sqrt{c}|| - x \oplus_{\mathbb{M}}^c y||) \frac{-x \oplus_{\mathbb{M}}^c y}{|| - x \oplus_{\mathbb{M}}^c y||}$$

Observe that we obtain the Euclidean counterparts in the limit: $\exp_x^c(v) \xrightarrow{c\to 0} x + v$ and $\log_x^c(v) \xrightarrow{c\to 0} y - x$.

D.3 Spherical Space and its Gyrostructure

We would like to work in a model of constant positive curvature that would recover the Euclidean space operations when the curvature goes to 0. Unfortunately, the parameterization of standard spherical geometry given by $\mathbb{S}_c^d = \{x \in \mathbb{R}^{d+1} \mid c\|x\|_2 = 1\}$ does not give achieve this goal. As a consequence, we will work in a different spherical model (still isometric with the standard one).

D.3.1 Stereographic Projection Model of the Sphere

In the following we construct a model by isometrically projecting the $\mathbb{S}_c^d$ to $\mathbb{R}^d$ (in perfect duality to the construction of the Poincaré model which is the isometric projection of the Lorentz model).
Fix the south pole $z = (0, -\frac{1}{\sqrt{c}})$. The **stereographic projection** is then the map:

$$\Phi : \mathbb{S}_c^d \to \mathbb{R}^d, x' \mapsto x = \frac{1}{1 + \sqrt{c}x'_{d+1}} x'_{1:d}$$

with the inverse given by

$$\Phi^{-1} : \mathbb{R}^d \to \mathbb{S}_c^d, x \mapsto x' = \left(\eta_x x, \frac{1}{\sqrt{c}}(\eta_x - 1) \right)$$

where we define $\eta_x = \frac{2}{1+c||x||^2}$.
Again we take the image of the sphere $\mathbb{S}_R^d$ under the extended projection $\Phi((0, \ldots, 0, -\frac{1}{c})) = 0$, leading to the stereographic model of the sphere. The metric tensor transforms as Janson (2015):

$$g_{ij}^{\mathbb{G}} = \eta_x^2 \delta_{ij}$$

Notice that the metric tensor of the Poincaré model and the stereographic model agree exactly if written in terms of the curvature K and not its absolute value c. The metric tensor in turn induces the distance between $x, y \in \mathbb{R}^d$ Janson (2015):

$$d_{\mathbb{G}}^c(x,y) = \frac{1}{\sqrt{c}} \cos^{-1}\left(1 - \frac{\frac{2}{c}||x-y||^2}{(\frac{1}{c}+||x||^2)(\frac{1}{c}+||y||^2)}\right)$$

This model is smoothly deformed to the Euclidean model in the curvature limit as $d_{\mathbb{G}}^c(x,y) \xrightarrow{c \to 0} 2\|x-y\|$. Moreover, replacing $c \mapsto -c$ with the identification $\sqrt{-c} = i\sqrt{c}$ and the application of Euler's formula $e^{ix} = \cos(x) + i\sin(x)$, recovers exactly the Poincaré distance.

D.3.2 Gyrovector Space in the Stereographic Model

As in the Poincaré model, we can again define an addition in the stereographic model:

Definition D.5. *For $x, y \in \mathbb{R}^d$, the **stereographic addition** is given by:*

$$x \oplus_{\mathbb{G}}^c y = \frac{(1-2cx^Ty - c||y||^2)x + (1+c||x||^2)y}{1-2cx^Ty + c^2||x||^2||y||^2} \in \mathbb{R}^d$$

Notice that as desired, $x \oplus_{\mathbb{G}}^c y \xrightarrow{c \to 0} x + y$, recovering Euclidean addition. Moreover, replacing $c \mapsto -c$ recovers Möbius addition. Actually, one can write both additions in a unified function of the sectional curvature K as

$$x \oplus^K y = \frac{(1-2Kx^Ty - K||y||^2)x + (1+K||x||^2)y}{1-2Kx^Ty + K^2||x||^2||y||^2}$$

Inserting $K = -c$ leads to Möbius addition, whereas $K = c$ results in the steregraphic addition. We hence have an addition that interpolates smoothly between all three geometries.

Moreover we can define a scalar multiplication:

Definition D.6. *For $s \in \mathbb{R}$ and $x \in \mathbb{R}^d$, the **scalar multiplication** in the stereographic model is given by:*

$$s \otimes_{\mathbb{G}}^c x = \frac{1}{\sqrt{c}} \tan\left(s \tan^{-1}(\sqrt{c}||x||)\right) \frac{x}{||x||} \in \mathbb{R}^d$$

Notice that we have a perfect correspondence as well by using Euler's identity and the formulas $\tan(ix) = i\tanh(x)$ and $\tan^{-1}(ix) = i\tanh^{-1}(x)$.

Furthermore, we state the following theorem:

Theorem 6. *The triple $(\mathbb{R}^d, \oplus_{\mathbb{G}}^c, \otimes_{\mathbb{G}}^c)$ equipped with the metric tensor $g_x = \eta_x^2 \mathbf{I}$ defines a gyrovector space.*

Proof is by algebraic verification of the identities in definition D.2.

In the following, we will introduce the statements dual to the ones in the Poincaré ball and obtain formulas for the exponential and logarithmic map. As in Lemma 3, we can rewrite the distance formula more elegantly:

Lemma 6. *For $x, y \in \mathbb{R}^d$, the distance $d_{\mathbb{G}}^c(x,y)$ can be written as:*

$$d_{\mathbb{G}}^c(x,y) = \frac{2}{\sqrt{c}} \tan^{-1}\left(\sqrt{c}||-x \oplus_{\mathbb{G}}^c y||\right)$$

Corollary 6.1. *For $x, y, z \in \mathbb{R}^d$ and $n \in \mathbb{N}$, it holds:*

- $d_{\mathbb{G}}^c(z \oplus_{\mathbb{G}}^c x, z \oplus_{\mathbb{G}}^c y) = d_{\mathbb{G}}^c(x,y)$
- $||r \otimes_{\mathbb{G}}^c x||_{\mathbb{G}} = |r|||x||_{\mathbb{G}}$ *where* $||x||_{\mathbb{G}} = d_{\mathbb{G}}^c(x, 0)$
- $n \otimes_{\mathbb{G}}^c x = x \oplus_{\mathbb{G}}^c \ldots \oplus_{\mathbb{G}}^c x$ *where the addition is performed n times.*

Notice how Corollary 6.1 is the exact analog to Corollary 5.1, again showing how natural the defined operations interact with the geometry.

For the exponential map, we first derive an expression using the Egregium theorem and the known formulas for unit speed geodesics on the sphere $\mathbb{S}_c^d$. This leads to a closed formula not involving any stereographic operations. We then proceed to rewrite this expression in terms of $\oplus_{\mathbb{G}}^c$, leading to a term that is easily invertible and hence providing us with the logarithmic map.

Corollary 6.2. *For $x \in \mathbb{R}^d$ and $v \in T_x\mathbb{R}^d$ the **exponential map** can be recast as*

$$\exp_x(v) = x \oplus_{\mathbb{G}}^c \left(\tan\left(\frac{\sqrt{c}\eta_x ||v||}{2}\right)\frac{v}{\sqrt{c}||v||}\right)$$

*Moreover for $y \neq x$, $\log_x : \mathbb{R}^d \to T_x\mathbb{R}^d$ the **logarithmic map** is given by*

$$\log_x(y) = \frac{2}{\sqrt{c}\eta_x}\arctan\left(\sqrt{c}|| - x \oplus_{\mathbb{G}}^c y||\right)\frac{-x \oplus_{\mathbb{G}}^c y}{|| - x \oplus_{\mathbb{G}}^c y||}$$

As expected, $\exp_x^c(v) \xrightarrow{c \to 0} x + v$, converging to the Euclidean exponential map. For the proof we again use the same arguments as in the proof of theorem 6. Once more, observe the perfect correspondence between the Poincaré (Lemma 5) and the stereographic models.

E Hyperbolic Logits

The final element missing in the hyperbolic neural network is the logit layer, a neccessity for any classification task. We here use the formulation of Ganea et al. (2018b). Denote by $\{1, \dots, K\}$ the possible labels and let $a_k \in \mathbb{R}^d$, $b_k \in \mathbb{R}$ and $x \in \mathbb{R}^d$. The output of a feed forward neural network for classification tasks is usually of the form

$$p(y = k|x) = \text{softmax}(\langle a_k, x\rangle - b_k)$$

In order to generalize this expression to hyperbolic space, the authors of Ganea et al. (2018b) realized that the term in the softmax can be rewritten as

$$\langle a_k, x\rangle - b_k = \text{sign}(\langle a_k, x\rangle - b_k)||a_k||d(x, H_{a_k,b_k})$$

where $H_{a,b} = \{x \in \mathbb{R}^d : \langle x, a\rangle - b\} = \{x \in \mathbb{R}^d : \langle -p + x, a\rangle\} = \tilde{H}_{a,p}$ with $p \in \mathbb{R}^d$.
As a first step, they define the hyperbolic hyperplane as

$$\tilde{H}_{a,p}^c = \{x \in \mathbb{P}_c^d : \langle -p \oplus_{\mathbb{M}}^c x, a\rangle\}$$

where now $a \in \mathcal{T}_p\mathbb{P}_c^d$ and $p \in \mathbb{P}_c^d$. They then proceed with the following lemma:

Lemma 7. $d_c^{\mathbb{M}}(x, \tilde{H}_{a,p}) = \frac{1}{\sqrt{c}}\sinh^{-1}\left(\frac{2\sqrt{c}|\langle -p \oplus_{\mathbb{M}}^c x, a\rangle|}{(1 - c|| - p \oplus_{\mathbb{M}}^c x||^2)||a||}\right)$

Using this equation, they were able to obtain the following expression for the logit layer:

Definition E.1. *The logits in hyperbolic space are defined as*

$$p(y = k|x) = \textit{softmax}\left(\frac{||a_k||}{\sqrt{c}}\sinh^{-1}\left(\frac{2\sqrt{c}\langle -p_k \oplus_{\mathbb{M}}^c x, a_k\rangle}{(1 - c|| - p_k \oplus_{\mathbb{M}}^c x||^2)||a_k||}\right)\right)$$

where $a_k \in \mathcal{T}_0\mathbb{P}_c^d \cong \mathbb{R}^d$, $x \in \mathbb{P}_c^d$ and $p_k \in \mathbb{P}_c^d$.

Combining all these operations leads to the definition of a hyperbolic feed forward neural network. Notice that the weight matrices W and the normal vectors a_k live in Euclidean space and hence can be optimized by standard methods such as ADAM Kingma & Ba (2015).

F GCNs in Product of Constant Curvature Spaces

Inspired by Gu et al. (2019), we can now take the model one step further and study graph neural networks in a product of spaces consisting of spherical and hyperbolic components.

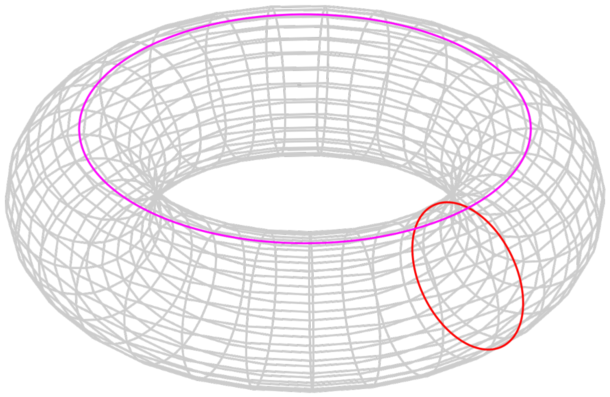

Figure 9: Intuition as to why $\mathbb{S}^1 \times \mathbb{S}^1 \ncong \mathbb{S}^2$

More precisely, we study spaces of the form

$$\mathcal{X} = \mathbb{M}^{d_1}_{c_1^-} \times \cdots \times \mathbb{M}^{d_m}_{c_m^-} \times \mathbb{G}^{b_1}_{c_1^+} \times \cdots \times \mathbb{G}^{b_l}_{c_l^+}$$

where we denote by $\mathbb{M}^d_{c^-}$ the Poincaré representation of $\mathbb{H}^d_{c^-}$ and by $\mathbb{G}^b_{c^+}$ the stereographic representation of $\mathbb{S}^b_{c^+}$.
For $v \in \mathcal{X}$, write

$$v = (v^1, \dots, v^m, \tilde{v}^1, \dots, \tilde{v}^l)$$

where each $v^i \in \mathbb{M}^{d_i}$ and $\tilde{v}^i \in \mathbb{G}^{b_i}$ represents a hyperbolic and spherical component.
It can be shown that $\mathcal{X}$ is again a Riemannian manifold equipped with a distance function, an exponential map and a logarithmic map decomposing as follows Ficken (1939); Turaga & Srivastava (2016):

Lemma 8. *For $x, y \in \mathcal{X}$, the distance between the two points is given by:*

$$d^2_{\mathcal{X}}(x, y) = \sum_{i=1}^{m} d^2_{\mathbb{M}^{d_i}}(x^i, y^i) + \sum_{i=1}^{l} d^2_{\mathbb{G}^{b_i}}(\tilde{x^i}, \tilde{y^i})$$

Moreover for $v \in \mathcal{T}_x\mathcal{X}$, the exponential map $\exp_x : \mathcal{T}_x\mathcal{X} \to \mathcal{X}$ decomposes as

$$\exp^{\mathcal{X}}_x(v) = (\exp^{\mathbb{M}^{d_1}}_{x_1}(v^1), \dots, \exp^{\mathbb{M}^{d_m}}_{x_m}(v^m), \exp^{\mathbb{G}^{b_1}}_{\tilde{x}^1}(\tilde{v}^1), \dots, \exp^{\mathbb{G}^{b_l}}_{\tilde{x}^l}(\tilde{v}^l))$$

Finally, for $y \in \mathcal{X}$, the logarithmic map $\log_x : \mathcal{X} \to \mathcal{T}_x\mathcal{X}$ takes the form

$$\log^{\mathcal{X}}_x(y) = \left(\log^{\mathbb{M}^{d_1}}_{x^1}(y^1), \dots, \log^{\mathbb{M}^{d_m}}_{x^m}(y^m), \log^{\mathbb{G}^{b_1}}_{\tilde{x}^1}(\tilde{y}^1), \dots, \log^{\mathbb{G}^{b_l}}_{\tilde{x}^l}(\tilde{y}^l)\right)$$

One may ask why we are also splitting the hyperbolic and spherical components into products. After all, for Euclidean components it is a well-known fact that for any $n, m \in \mathbb{N}$, we have that $\mathbb{R}^n \times \mathbb{R}^m = \mathbb{R}^{n+m}$. This property does not hold anymore for both spherical and hyperbolic products of spaces. Consider for instance $\mathbb{S}^1 \times \mathbb{S}^1$. It turns out that this product of two circles is not isomorphic to the 2-sphere $\mathbb{S}^2$ but rather two the torus $\mathbb{T}^2$. The reader may check this statement by calculating the fundamental groups of both $\mathbb{S}^1 \times \mathbb{S}^1$ and $\mathbb{S}^2$ and observe that they are not identical.
Thus splitting both hyperbolic and spherical components leads to a more general class of models.

F.1 Gyrostructure in Products of Gyrovector spaces

It turns out that we can again equip $\mathcal{X}$ with a gyrovector space structure, inheriting the structure from all the smaller components:

Theorem 7. *For $x, y \in \mathcal{X}$, define the addition $\oplus_{\mathcal{X}}$ and the scalar multiplication $\otimes_{\mathcal{X}}$ as:*

- $x \oplus_{\mathcal{X}} y = (x^1 \oplus_{\mathbb{M}^{d_1}} y^1, \dots, x^m \oplus_{\mathbb{M}^{d_m}} y^m, \tilde{x}^1 \oplus_{\mathbb{G}^{b_1}} \tilde{y}^1, \dots, \tilde{x}^l \oplus_{\mathbb{G}^{b_l}} \tilde{y}^l)$

- $r \otimes_{\mathcal{X}} x = (r \otimes_{\mathbb{M}^{d_1}} x^1, \dots, r \otimes_{\mathbb{M}^{d_m}} x^m, r \otimes_{\mathbb{G}^{d_1}} \tilde{x}^1, \dots, r \otimes_{\mathbb{G}^{d_l}} \tilde{x}^m)$

Moreover for the norms $||x||$ and $||y||$ where $x, y \in \mathcal{X}$ we define:

- $||x|| \oplus_{\mathcal{X}} ||y|| = \sqrt{\sum_{i=1}^{m}(||x^i|| \oplus_{\mathcal{X}^i} ||y^i||)^2 + \sum_{i=1}^{l} ||\tilde{x}^i|| \oplus_{\mathcal{X}}^{i} ||\tilde{y}^i||)^2}$
- $r \otimes_{\mathcal{X}} ||x|| = \sqrt{\sum_{i=1}^{m}(r \otimes_{\mathcal{X}^i} ||x^i||)^2 + \sum_{i=1}^{l}(r \otimes_{\mathcal{X}^i} ||\tilde{x}^i||)^2}$

Then the triple $(\mathcal{X}, \oplus_{\mathcal{X}}, \otimes_{\mathcal{X}})$ defines a gyrovector space. Moreover, we have the following two invariances:

- $||r \otimes_{\mathcal{X}} x||_{\mathcal{X}} = d_{\mathcal{X}}(r \otimes_{\mathcal{X}} x, 0) = r||x||_{\mathcal{X}}$
- $d_{\mathcal{X}}(b \oplus_{\mathcal{X}} x, b \oplus_{\mathcal{X}} y) = d_{\mathcal{X}}(x, y)$

The gyrovector space over $\mathcal{X}$ allows us again to extend all the quantities derived in the previous chapter to the more general framework of products of Riemannian spaces of constant sectional curvature.

F.2 Right Matrix Multiplication in Product of Spaces

Given an embedding $X \in \mathbb{R}^{n \times (d_1 + \dots + d_m + b_1 + \dots + b_l)}$ where each row $X_{i\bullet} \in \mathcal{X}$ and a weight matrix $W \in \mathbb{R}^{(d_1 + \dots + d_m + b_1 + \dots + b_l) \times (e_1 + \dots + e_m + f_1 + \dots + f_l)}$, we want to define again the notion of a right matrix multiplication, leading to a mixing of the different features. First express the embedding through its different components:

$$X = \begin{bmatrix} X^1_{1\bullet} & \dots & X^m_{1\bullet} & \tilde{X}^1_{1\bullet} & \dots & \tilde{X}^l_{1\bullet} \\ & \vdots & & & & \\ X^1_{n\bullet} & \dots & X^m_{n\bullet} & \tilde{X}^1_{1\bullet} & \dots & \tilde{X}^l_{1\bullet} \end{bmatrix} = [X^1, \dots, X^m, \tilde{X}^1, \dots, \tilde{X}^l]$$

where $X^j_{i\bullet} \in \mathbb{B}^{d_j}_{c_j^-}$, $\tilde{x}^j_{i\bullet} \in \mathbb{B}^{b_j}_{c_j^+}$, $X^i \in \mathbb{R}^{n \times d_i}$ and $\tilde{X}^i \in \mathbb{R}^{n \times b_i}$. Moreover, express the weight matrix W as

$$W = \begin{bmatrix} W^1_{\bullet 1} & W^1_{\bullet 2} & \dots & w^1_{\bullet n} \\ \vdots & \vdots & & \vdots \\ W^m_{\bullet 1} & W^m_{\bullet 2} & \dots & W^m_{\bullet n} \\ \tilde{W}^1_{\bullet 1} & \tilde{W}^1_{\bullet 2} & \dots & \tilde{W}^1_{\bullet n} \\ \vdots & \vdots & & \vdots \\ \tilde{W}^l_{\bullet 1} & \tilde{W}^l_{\bullet 2} & \dots & \tilde{W}^l_{\bullet n} \end{bmatrix} = \begin{bmatrix} W^1 \\ \vdots \\ W^m \\ \tilde{W}^1 \\ \vdots \\ \tilde{W}^l \end{bmatrix}$$

where $W^i \in \mathbb{R}^{d_i \times (e_1 + \dots + e_m)}$ and $\tilde{W}^i \in \mathbb{R}^{b_i \times (f_1 + \dots + f_l)}$.

Definition F.1. *The right matrix multiplication in the product of Riemannian manifolds of constant sectional curvatures $\mathcal{X}$ is given by*

$$X \otimes_{\mathcal{X}} W = \exp_0^{\mathcal{X}}(\log_0^{\mathcal{X}}(X)W)$$

Let us examine the behaviour of $\otimes_{\mathcal{X}}$ a bit more closely using the notation introduced above.

We first look at the inner term of the matrix multiplication:

$$\begin{aligned} Z &= \left(\log_0^{\mathbb{M}^{d_1}}(X^1), \dots, \log_0^{\mathbb{G}^{d_m}}(X^m),\ \log_0^{\mathbb{G}^{b_1}}(\tilde{X}^1), \dots, \log_0^{\mathbb{G}^{b_l}}(\tilde{X}^l)\right) \\ &= \left(Z^1, \dots, Z^m,\ \tilde{Z}^1, \dots, \tilde{Z}^l\right) \end{aligned}$$

Now the right multiplication with W can be expressed as the sum over the smaller matrices:

$$ZW = Z^1W^1 + \dots + Z^mW^m + \tilde{Z}^1\tilde{W}^1 + \dots + \tilde{Z}^l\tilde{W}^l \in \mathbb{R}^{n \times (e_1 + \dots + e_m + f_1 + \dots f_l)}$$

Notice that here a mixing of the different feature components is happening in the respective tangent spaces at zero. Hence this operation is not simply decomposing and the different components interact with each other. This interaction is then mapped back to the product manifold using the exponential map.

Applications of non-linearities can also again be defined using exponential and logarithmic maps:

Definition F.2. *Given any map $\sigma : \mathbb{R}^{d_1+\cdots+d_m+b_1+\cdots+b_l} \to \mathbb{R}^{d_1+\cdots+d_m+b_1+\cdots+b_l}$, $x \mapsto \sigma(x)$, we define its product space dual as*

$$\sigma^{\mathcal{X}} : \mathcal{X} \to \mathcal{X},\ x \mapsto \exp_0^{\mathcal{X}}(\sigma(\log_0^{\mathcal{X}}(x)))$$

F.3 Product Logits

One could be tempted to extend the logits through a combination of the ideas from this section and the previous one. More precisely, one could try to define a hyperplane as

$$\tilde{H}^c_{a,p} = \{x \in \mathcal{X} : \langle -p \oplus_{\mathcal{X}} x, a\rangle\ = 0\}$$

Then one could again leverage the component-wise nature of $\oplus_{\mathcal{X}}$ and the linearity of $\langle\rangle$ in order to arrive at an expression decomposing over the components:

$$\tilde{H}^c_{a,p} = \{x \in \mathcal{X} : \sum_{i=1}^{m}\langle -p^i \oplus_{\mathbb{M}}^{c_i^-} x^i, a^i\rangle\ +\ \sum_{i=1}^{k}\langle -\tilde{p}^i \oplus_{\mathbb{G}}^{c_i^+} \tilde{x}^i, \tilde{a}^i\rangle\ = 0\}$$

This expression is unfortunately not easy to handle as one cannot write this as the product of the smaller hyperplanes in each component. Especially obtaining the minimal distance of a point $x \in \mathcal{X}$ to this hyperplane is not a simple task. Fortunately, it turns out that there is an easier approach:

Definition F.3. *The* ***logits*** *in the product of Riemannian manifold of constant sectional curvature are defined as*

$$p(y = k|x; a, p) = \sum_{i=1}^{m} p_{\mathbb{M}^{d_i}_{c_i^-}}(y^i = k|x^i; a^i, p^i) + \sum_{i=1}^{k} p_{\mathbb{G}^{b_i}_{c_i^+}}(\tilde{y}^i = k|\tilde{x}^i; \tilde{a}^i, \tilde{p}^i)$$

where $a \in \mathcal{T}_x\mathcal{X} \cong \mathcal{T}_{x^1}\mathbb{B}^{d_1}_{c_1^-} \times \cdots \times \mathcal{T}_{x^m}\mathbb{B}^{d_m}_{c_m^-} \times \mathcal{T}_{\tilde{x}^1}\mathbb{R}^{b_1} \times \cdots \times \mathcal{T}_{\tilde{x}^m}\mathbb{R}^{b_m}$ and $p \in \mathcal{X}$.

Notice that in the limit where all curvatures c_i^- and $\tilde{c}_i^+$ go to zero, we recover the expression

$$\sum_{i=1}^{m} 4\langle -p^i + x^i, a^i\rangle + \sum_{i=1}^{l} 4\langle -\tilde{p}^i\ +\ \tilde{x}^i, \tilde{a}^i\rangle = 4\langle -p + x, a\rangle$$

which are exactly the logits in the Euclidean space $\mathbb{R}^{d_1+\cdots+d_m+b_1+\cdots+b_l}$ scaled by the factor 4.
We hence again recover the Euclidean counterpart when using Definition F.3 for the logits.

F.4 Left Matrix Multiplication and Midpoints

Again, the missing ingredient for the definition of a left matrix multiplication is the notion of taking an average. As we will see in the following, the gyromidpoint can be easily extended component-wise while still preserving the desirable properties of its one-component counterpart.

Definition F.4. *For $\{x_1, \ldots, x_n\} \subset \mathcal{X}$ and weights $\{\alpha_1, \ldots, \alpha_n\} \subset \mathbb{R}$ the* ***weighted gyromidpoint*** *in the product manifold model is given by*

$$m_{\mathcal{X}}(x_1, \ldots x_n; \alpha_1, \ldots, \alpha_n) = \Big(m_{\mathbb{M}^{d_1}_{c_1^-}}(x_1^1, \ldots x_n^1; \alpha_1, \ldots, \alpha_n), \ldots, m_{\mathbb{M}^{d_m}_{c_m^-}}(x_1^m, \ldots x_n^m; \alpha_1, \ldots, \alpha_n),$$
$$m_{\mathbb{G}^{b_1}_{c_1^+}}(\tilde{x}_1^1, \ldots \tilde{x}_n^1; \alpha_1, \ldots, \alpha_n), \ldots, m_{\mathbb{G}^{b_l}_{c_l^+}}(\tilde{x}_1^l, \ldots \tilde{x}_n^l; \alpha_1, \ldots, \alpha_n)\Big)$$

We recover very similar properties to the Poincaré and spherical gyromidpoint:

Lemma 9. *For $\{x_1, \ldots, x_n\} \subset \mathcal{X}$, $z \in \mathcal{X}$ and weights $\{\alpha_1, \ldots, \alpha_n\} \subset \mathbb{R}$ it holds that:*

- $m_{\mathcal{X}}(z \oplus_{\mathcal{X}} x_1, \ldots, z \oplus_{\mathcal{X}} x_n;\ \alpha_1, \ldots, \alpha_n) = z \oplus_{\mathcal{X}} m_{\mathcal{X}}(x_1, \ldots, x_n;\ \alpha_1, \ldots, \alpha_n)$
- $d_{\mathcal{X}}(x_1, m^c_{\mathcal{X}}(x_1, x_2; \frac{1}{2}, \frac{1}{2})) = \frac{1}{2} d^c_{\mathcal{X}}(x_1, x_2)$
- $m_{\mathcal{X}}(x_1, \ldots, x_n; \alpha_1, \ldots \alpha_n) \xrightarrow{c_1^-, \ldots, c_m^-, c_1^+, \ldots, c_l^+ \to 0} m_{\mathbb{E}}(x_1, \ldots, x_n; \alpha_1, \ldots \alpha_n)$

Being already equipped with a scalar multiplication, we are thus ready to introduce the left matrix multiplication:

Definition F.5. *Given $X \in \mathbb{R}^{n \times (d_1 + \cdots + d_m + b_1 + \cdots + b_l)}$ with rows $X_{i\bullet} \in \mathcal{X}$ and $A \in \mathbb{R}^{n \times n}$, the* ***product left matrix multiplication*** *$Z = A \otimes_{\mathcal{X}} X$ is defined row-wise via*

$$Z_{i\bullet} = (\sum_{j=1}^{n} A_{ij}) \otimes_{\mathcal{X}} m_{\mathcal{X}}(X_{1\bullet}, \ldots, X_{n\bullet}; A_{i1}, \ldots A_{in})$$

Moreover, Z can be written in terms of the left matrix multiplications of its components:

$$Z = (A \otimes_{\mathbb{M}^{d_1}_{c_1^-}} X^1, \ldots, A \otimes_{\mathbb{M}^{d_m}_{c_m^-}} X^m, A \otimes_{\mathbb{G}^{b_1}_{c_1^+}} \tilde{X}^1, \ldots, A \otimes_{\mathbb{G}^{b_l}_{c_l^+}} \tilde{X}^l)$$

Notice how in contrast to the right matrix multiplication, the left matrix multiplication decomposes over the factors independently.

F.5 Product GCN

Having gathered all needed tools in the previous section, we can introduce the architecture of the product GCN.

Assume we are given a graph with node level features $G = (V, A, X)$ where $X \in \mathbb{R}^{n \times d}$ and adjacency $A \in \mathbb{R}^{n \times n}$. In order to map the Euclidean features X to the product manifold we apply the projection $X \mapsto \frac{X}{2\sqrt{c}||X||_{max}}$ component-wise for the hyperbolic components and the identity for the spherical components. Notice that we hence need to split the feature dimension into the separate components by writing $d = \sum_{i=1}^{k} d_i + \sum_{i=1}^{l} b_i$, therefore deciding in advance what features belong to what component.

We thus obtain a preprocessed first embedding $\tilde{X} = \text{proj}^{\mathcal{X}}(X) \in \mathbb{R}^{n \times (d_1 + \cdots + d_m + b_1 + \cdots + b_l)}$ where each row $\tilde{X}_{i\bullet} \in \mathcal{X}$ lives now in the product manifold.

For $l \in \{0, \ldots, L-2\}$, the $(l+1)$-th layer of the product GCN is given by

$$H^{(l+1)} = \sigma^{\mathcal{X}} \left(\hat{A} \otimes_{\mathcal{X}} \left(H^{(l)} \otimes_{\mathcal{X}} W^{(l)} \right) \right)$$

where $H^{(0)} = \tilde{X}$, σ is some non-linearity and $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$.
The final layer is a product logit layer:

$$H^{(L)} = \text{softmax} \left(\hat{A} \text{ logit} \left(H^{(L-1)}, W^{(L-1)} \right) \right)$$

Notice how this architecture generalizes $\mathbb{H}$-GCN and $\mathbb{S}$-GCN since we obtain those by only using one hyperbolic or spherical component respectively.

G Experimental Details

We here present training details for the node classification experiments.

We closely follow the training and evaluation scheme from previous work, e.g. Klicpera et al. (2019). We split the data into training, early stopping, validation and test set. Namely we first split the dataset into a known subset of size n_{known} and an unknown subset consisting of the rest of the nodes. For all the graphs we use $n_{known} = 1500$ except for MS Academics, where we use $n_{known} = 5000$.

Dataset	Type	Classes	Features	Nodes	Edges	Label rate	Avg sp
Citeseer	Citation	6	3703	2110	3668	0.036	9.31
Cora-ML	Citation	7	2879	2810	7981	0.047	5.27
Pubmed	Citation	3	500	19717	44324	0.003	6.34
MS-Academic	Co-author	15	6805	18333	81894	0.0016	5.34

Table 3: Summary statistics for the four datasets, where sp denotes shortest path.

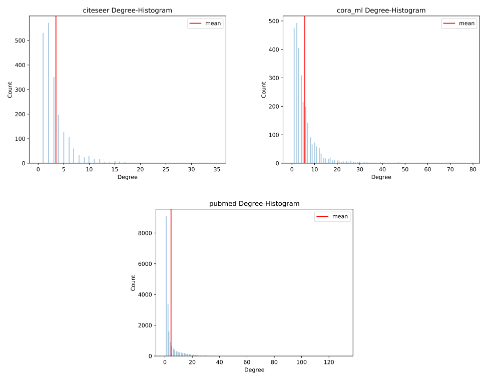

Figure 10: Histogram of node degrees

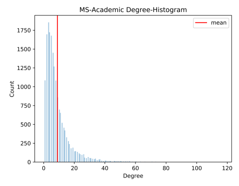

Figure 11: Histogram of node degrees

The known subset is further split into a training set consisting of 20 data points per label, an early stopping set of size 500 and a validation set of the remaining nodes. Notice that the whole structure of the graph and all the node features are used in an unsupervised fashion since the embedding of a training node might for instance depend on the embedding of a node from the validation set. But when calculating the loss, we only provide supervision with the training data.
The unknown subset serves as the test data and is only used for the final evaluation of the model. Hyperparameter-tuning is performed on the validation set. We further use early stopping in all the experiments. We stop training as soon as the early stopping cross entropy loss has not decreased in the last $n_{patience} = 200$ epochs or as soon as we have reached $n_{max} = 2000$ epochs. The model chosen is the one with the highest accuracy score on the early stopping set. For the final evaluation we test the model on 10 different data splits and report mean accuracy and bootstrapped confidence intervals. We use the described setup for both the Euclidean and non-Euclidean models to ensure a fair comparison.

Training details for the Euclidean Baselines We trained the Euclidean models with the hyperparameters chosen as reported in Klicpera et al. (2019). Namely, for GCN we use one hidden layer of size 64, dropout on the embeddings and the adjacency of rate 0.5 as well as L^2-regularization for the weights of the first layer with $\lambda = 0.02$. Only for Cora-ML we had to adjust the regularization factor λ to 0.002 to ensure similar scores as achieved in Klicpera et al. (2019).

Summary of training details. All Non-Euclidean models use biased-L2 regularization with $\alpha = 10$ and $\lambda = 2e-2$. Euclidean models used L2 regularization with the same parameter λ. We used a combination of dropout and dropconnect for the non-Euclidean models. All models have the same number of parameters. We use 2 GCN layers, hidden dimension 64. Product models split hidden dimension into [32, 32] and also input features equally. Non-Euclidean models do not use additional non-linearities. Euclidean parameters use a learning rate of 0.01 for all models using ADAM. The curvatures are learned using gradient descent with lr 0.01 as well. We use early stopping: we first train for a maximum of 2000 epochs, then we check every 200 epochs for improvement in the validation cross entropy loss; if that is not observed, we stop.

Learned curvatures .
Citeseer:
Hyp GCN: Trained curvature, average curvature over all runs: -1.057 +-0.03
Sphr GCN: Trained curvature, average curvature over all runs: 0.951 +-0.019
Prod GCN: Trained curvatures, average curvatures: [1.331, -0.91]

Cora
Hyp GCN: Trained curvature, average curvature: -1.127 +-0.011
Sphr GCN: Trained curvature, average curvature: 0.857+-0.013
Prod GCN: Trained curvature, average curvature: [-1.03, -1.01]

Pubmed:
Hyp GCN: Trained curvature, average curvature: 1.123 +- 0.01
Sphr GCN: Trained curvature, average curvature: 0.896 +- 0.008
Prod GCN: Not training curvature, fixed to [-1, -1]

Ms-Academic
Hyp GCN: Trained curvature, average curvature: 1.26 +- 0.09
Sphr GCN: Trained curvature, average curvature: 0.8 +- 0.07
Prod GCN: Not training curvature, fixed to [-1, -1]

H Graph Curvature Estimation Algorithm

1. Fix a node $m \in G$ and sample two neighbouring nodes $a, b \in G$ uniformly. Further sample an additional reference node $c \in G$ uniformly (again avoiding $m = c$).
2. Calculate $\psi(m; a, b; c) = \frac{1}{2d_G(a,b)} \left(d_G^2(a, m) + \frac{d_G^2(b,c)}{4} - \left(\frac{d_G^2(a,b)+d_G^2(a,c)}{2} \right) \right)$
3. Reiterate the above sampling n_{iter} times and obtain an average curvature at node m.
4. Do this procedure for every node $m \in G$.