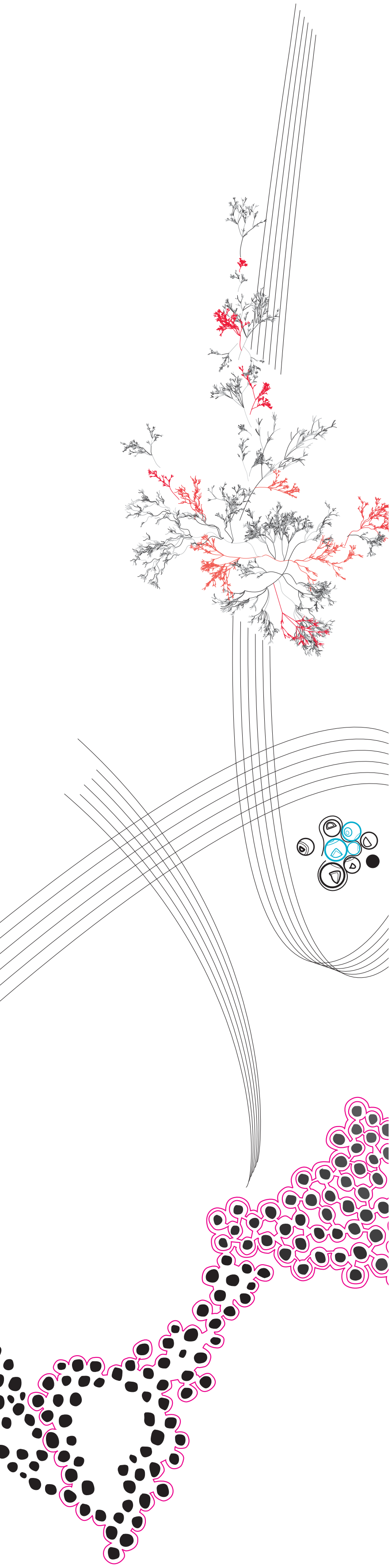
MSc Thesis Applied Mathematics



Understanding the Performance of Hyperbolic Graph Neural Networks

B.P. Petrov

Supervisors:
prof. dr. C. Brune
dr. C. Stegehuis

Committee member:
dr. Annika Betken

September, 2023

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Abstract

Hyperbolic graph neural networks (HGNNs) have been gaining prominence in the field of machine learning. They have been shown to perform better than their Euclidian counterparts on a wide variety of datasets with underlying hyperbolic geometry [1]. However, a persisting problem of HGNNs is the lack of understanding of what exactly causes this performance gap, and thus in which specific cases their usage is recommended [2]. In this thesis we investigate the correlation between dataset properties and HGNN performance, in particular by taking HGCN [1] and GCN [3] as comparable models. Experiments show that δ-hyperbolicity, specifically δ_{avg} is a reliable indicator that HGCN will perform better than GCN on the given dataset, except when the dataset used is very dense. In the latter case the degree distribution should be considered, and other properties like clustering or Balanced Forman curvature do not give a good indication. Furthermore, our experiments show that, when the underlying hyperbolic geometry is present, the HGCN is good at handling noise in the graph structure, whereas the GCN benefits when noise in the features is present. The role of the learnable curvature of the HGCN is investigated and appears to likely not relate directly to curvature in the data. Lastly, experiments show that, when the given dataset is highly hyperbolic, the HGCN does very well in case the embedding dimension is low and does not benefit greatly from a higher embedding dimension. The GCN performs poorly when the dimension is low and improves steadily when the dimension increases. On the other hand, when the dataset is not hyperbolic, both models perform very similar, but the HGCN still maintains a slight edge when the embedding dimension is low.

Contents

1 Introduction

Graph neural networks (GNNs) have in recent years become an important part of the AI field, as they have been shown to be great tools for learning on graph data [4]. Their areas of application range from recommender systems for movies to analysis of transportation networks to predicting protein-protein interaction in the human body. This diversity comes from the fact that many real-world phenomena can be described as graphs, creating many tasks for graph learning. And there is an equally diverse library of model architectures that can be used for these tasks, just as there is a huge variety of models in deep learning as a whole [5]. This variety often leads to the common problem: which neural network architecture should be chosen for a specific context?

Quite recently, GNNs have been combined with another recently growing field within AI: geometric deep learning [6]. In this field, the goal is to create models that suit the underlying geometry of the data, whatever that may be. This could for example be spherical data (such as the omnidirectional vision of drones [7]), or hyperbolic data. Hyperbolic deep neural networks allow for the embedding of data with hierarchical structure into few dimensions [8]. The first neural network of this type was developed in 2018 in the form of Hyperbolic Neural Networks [9], though there had been previous works utilizing hyperbolic geometry in the context of machine learning [8]. Soon after, hyperbolic learning made its way to GNNs, resulting in so-called hyperbolic graph neural networks, or HGNNs [2] (in this work we use HGNN to describe the general framework hyperbolic GNNs, as opposed to Hyperbolic Graph Neural Networks [10], a paper describing a particular HGNN architecture published in 2019).

The motivation for the use of HGNNs is straightforward: GNNs typically work by embedding the nodes of a graph in Euclidian space. However, this poses problems when working with graph data with underlying hyperbolic geometry, meaning hierarchical or tree-like structure [1]. The intuition behind this is the following: consider the “volume” of a graph, which is number of nodes surrounding an arbitrary node of a graph, within a given distance/radius. For trees, this volume grows exponentially with the radius. In contrast, in Euclidian space a ball’s volume grows polynomially with the radius. This leads to high distortion in the embedding, and requires more dimensions to embed in. On the other hand, in hyperbolic space the volume increases exponentially with the radius, making it a good match for this problem. Thus, hyperbolic space is much better suited for embedding hyperbolic data, and, as it turns out, many real-world datasets exhibit properties of hyperbolic geometry [8]. Examples of this are human hierarchies, cell development processes, and citation networks.

Based on this knowledge, researchers began creating hyperbolic graph neural networks, with one of the first ones being Hyperbolic Graph Convolutional Networks (HGCN) [1] in 2019. HGCN is not the first model to make use hyperbolic space for learning on graphs, but it is the first such deep model and thus the first to capture node features in addition to the graph structure itself. All of this makes it an important milestone in the field. In the years since then, many other models have been created around the HGNN framework, such as Fully Hyperbolic GNN [11] and Hyperbolic-to-hyperbolic GNN [12], which improve on the technical aspects of HGNNs. Furthermore, HGNNs have been used in a variety of tasks like recommender systems [13] and algorithmic trading [14].

Despite the abundance of progress in HGNNs, there are several challenges that still persist. These are formulated by Yang et al. [2] in a survey of HGNN architectures. Challenge I: Real-life data often exhibits curvatures that are not constant but vary over the dataset, how can HGNNs leverage this? Challenge II: How can the learning process be defined in hyperbolic space? Challenge III: Why does hyperbolic space perform better in some cases? In which cases specifically are HGNNs guaranteed to perform better? Challenge IV: How can HGNNs be made more efficient as to be feasible for large-scale datasets?

In this work we address challenge III, meaning why and in which cases HGNNs are superior. Specifically we want to investigate in which cases HGNNs perform better than regular GNNs and why. Through this we also tackle the larger problem in the AI sphere which is how to choose right model depending on the context. Chami et al. [1] show that there are certain contexts where the HGCN performs better than (any) Euclidian graph learning methods and concludes that this performance advantage can be determined by Gromov’s δ-hyperbolicity. This is an important result but it leaves a lot of questions, for example: Is δ the only property that can predict this difference in performance? What is the role of the learnable curvature? How does the embedding dimension affect the performance?

This thesis investigates the following research questions (RQs) with respective subquestions:

RQ1: What properties of the dataset guarantee that HGNNs will perform better than (Euclidian) GNNs?

- Is the δ-hyperbolicity always a good indicator of this?
- Are there other dataset statistics that can predict this difference in performance?
- How does noise in the data affect the performance of HGNNs and GNNs?

RQ2: What role does the curvature play as a learnable parameter in the HGCN?

- Does the curvature converge during the optimization process?
- How does the curvature differ per layer in the neural network?
- If the learned curvature converges to a value, can this value be estimated by looking only at the data?

RQ3: Can the perfomance gap between HGNN and GNN be compensated for by increasing the embedding dimension of the GNN?

- Is the performance gap between the two models dependent on the embedding dimension?
- Can a high-dimension GNN embedding always perform better than a low-dimension HGNN embedding?

Having an answer to the first research question would make it easier to decide whether to use a hyperbolic model given only the dataset. The second research question could remove the need for setting the curvature as a learnable parameter in the HGCN, instead making it possible to set the value before training based on properties of the dataset. Finally, the third research question would further outline the use cases of hyperbolic and Euclidian models, when a given embedding dimension is required. All of these together serve to paint a paint a picture of what GNN model to use in different cases, as well as improving the understanding of HGNNs as a whole.

This thesis proceeds as follows: Section 2 starts with a broad collection of theoretical background needed to properly understand the HGNNs. This includes a review of learning on graphs where the basic functionality of GNNs is presented. Then hyperbolic space will be investigated: What is hyperbolic space and how can we work with it? How can we determine if a graph has underlying hyperbolic geometry? Section 3, the modelling section, goes in detail into the mechanisms of the HGCN. Furthermore, in this section the datasets to be used are introduced, along with several random graph models and dataset properties to be investigated. Section 4 describes the experiments to answer the previously stated research questions. The thesis ends with a discussion of the results in Section 5 and a conclusion in Section 6.

2 Background

The theoretical background of this thesis is split into two parts: learning on graphs, and hyperbolic geometry. These two topics provide the theoretical foundation to hyperbolic graph neural networks, which are presented as the last part of this section.

2.1 Learning on graphs

Learning on graphs is a wide field with many components. We briefly explain the basics and continue with the main focus of this thesis, graph neural networks. Basics of graphs in general and basics on deep learning are omitted. Furthermore, graph learning contains many advanced topics not addressed here, such as graph augmentation methods, generative methods, spectral GNNs, knowledge graphs, and more. This subsection is largely based on the “Machine Learning with Graphs” lectures by Jure Leskovec [15].

2.1.1 Basics

Many real-world contexts can be described using graphs, from which different applications arise. Molecules can be described as nodes being atoms and links being the bonds between them, with application to drug discovery as an example. Traffic networks with their locations and connections between them are also graphs, and can be used to predict traffic on a given route. Social networks, as in people and any kind of social relations between them, can be used to recommend new connections or predict attributes of a given person. In fact any scenario where there are entities that interact or relate with each other can be described as a graph where graph learning tasks can be formulated. Often scenarios where a graph describes a real-world phenomenon, also provide rich node features that can be used in graph learning methods.

In graph learning the goal is usually to embed the nodes of a graph in some space, while preserving the data structure. Preserving the structure means that nodes that are similarly connected in the graph are close to each other in the embedding space. As an example of a task, the "average" of two nodes in the graph is not defined, but embedding them in $\mathbb{R}^2$ makes finding the average trivial. The same holds for more elaborate processes such as node classification, a common task for graph learning. Suppose we want to classify some nodes of the graph, which we do by embedding them into $\mathbb{R}^2$ and performing standard multiclass classification there (Figure 1). If, for example, some of the labels in the left picture were unknown, we could use the right picture to predict their class. In this example our operations are purely based on the graph structure, with no extra information given by the nodes.

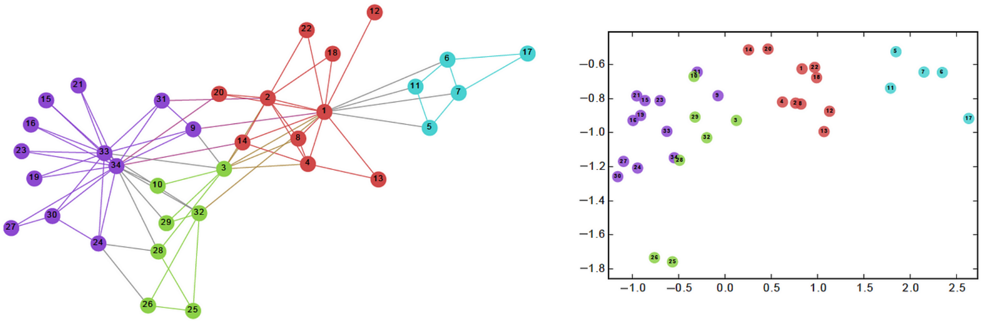


Figure 1: Example of embedding into 2 dimensions for node classification.

Another common task for graph learning is link prediction, meaning predicting between which nodes there are links. There are different options for this. First the nodes are embedded in some space, then, in order to predict whether a link exists between two given nodes, their embeddings are either concatenated or multiplied using dot product, followed again by any usual classifier. There are also more elaborate options, such as the Fermi-Dirac encoder, used in the HGCN [1]. Another possible task for graph leaning is graph classification, which can be done for example by averaging node embeddings, or using an artificial fully connected node. Other possible tasks exist, such as node clustering, graph generation, graph evolution, however these are not of interest to this thesis.

Methods for learning on graphs typically leverage two things: the graph structure (connections between nodes) and node features, which can be given in the data or induced from the graph itself. Learning on graphs can be classified in transductive learning (methods that do not generalize to unseen data) and inductive learning (methods that generalize to unseen data). The shallow methods described later in this chapter are always transductive, whereas the deep methods may be inductive.

2.1.2 Traditional features

Traditional features are inferred from the graph structure. These can be node-level features, link-level features, and graph-level features. Typical node-level features are degree, centrality (how important is a given node in the graph, for which different measures exist), clustering coefficient (how connected are each node's neighbors, or counting the number of triangles a node is part of), and graphlets (a generalization of clustering that counts structures other than triangles).

Link-level features are based on properties of pairs of nodes, and fall in three main categories: distance-based, local neighborhood overlap (distance up to 1), global neighborhood overlap (whole graph). Graph-level features are based on kernel methods, one simple example being the graphlet kernel: describe a graph as counts of graphlets it containts (bag-of-graphlets). Once the kernel is computed, getting the similarity of two graphs can simply be done by taking the dot product.

In more elaborate methods, such as graph neural networks, these traditional features are typically not directly used, as the neural network itself can infer them.

2.1.3 Shallow node embeddings

Node embeddings are a way of mapping the nodes of a graph to points in some space, where we can perform operations on them. This has to be done in a way that preserves the similarity of nodes: nodes that are close together and/or similar in the graph should be close to each other in the embedding.

The simplest node embeddings are the shallow ones. These are limited in that they do not work on unseen nodes (transductivity) and they do not capture structural similarity, they just compare neighborhoods. They are an unsupervised method thus they are task independent. They also do not take into account node features.

The main types of shallow embedding are: naive (based on adjacency in graph), neighborhood overlap, and random walk methods, such as DeepWalk [16] and Node2vec [17]. In general, different models might perform better or worse on different tasks so there isn't a "best" embedding method.

It is also possible to embed an entire graph. Ways to do this are using some node embedding and summing the embeddings, adding a virtual node that is connected to all other nodes, or by using an anonymous walk.

2.1.4 Graph neural networks

Graph Neural Networks (GNNs) are a deep node embedding method. They work using message passing: iteratively updating the representations of nodes by exchanging information with their neighbors. In this way, information is propagated through the graph in each layer of the neural network. There are three steps in message passing: feature transformation (1), neighborhood aggregation (2), and nonlinear activation (3) [1][2].

$$\text{Feature transformation} \qquad \mathbf{h}_i^\ell = W^\ell \mathbf{x}_i^{\ell-1} + \mathbf{b}^\ell \tag{1}$$

$$\text{Neighborhood aggregation} \qquad \mathbf{y}_i^\ell = \text{AGG}\left(\mathbf{h}^\ell\right)_i \tag{2}$$

$$\text{Nonlinear activation} \qquad \mathbf{x}_i^\ell = \sigma\left(\mathbf{y}_i^\ell\right) \tag{3}$$

Here ℓ is the layer to be computed, i is the index of the current node, W and $\mathbf{b}$ are the weights and bias of the linear layer respectively, σ is the nonlinear activation function, and AGG is the aggregation method. In fact, (1) and (3) are simply the steps that would be performed in a standard deep neural network, with the addition being (2), the aggregation step. This step is crucial as it allows the propagation of information within the graph, and it can differ greatly between different architectures. It must always be permutation invariant, as the neighbors of a node are unordered (unlike a standard convolution in for example image processing CNNs).

There are three main types of GNNs, determined by the aggregation mechanism [18]:

- Convolutional: weighed average of node features and node neighbor features, with predetermined weights (such as average, sum, or max).
- Attentional: same as convolutional, but weights are learned by a MLP where the input is the concatenated features of two nodes, and the output is the weight between them.
- General: a general nonlinear function computes the incoming features from each neighbor, which are then aggregated using standard convolution methods.

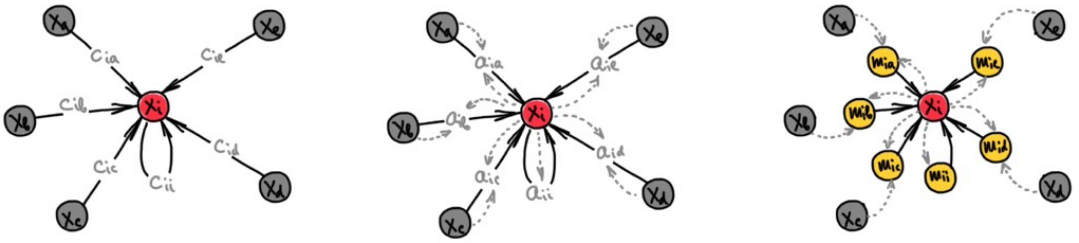


Figure 2: Message passing types, from left to right: convolutional, attentional, general [18].

The message passing mechanism determines the expressiveness of the model: general is more expressive than attentional, attentional more expressive than convolutional. Naturally, the more expressive models are also more computationally expensive, so a conscious choice of architecture always needs to be made with this kept in mind.

There are a few important points to be made here. First of all, (1) and (2) are not completely separate in the general message passing case. There, the features are transformed while taking neighbors into account. However, in this work we will focus on the convolutional and attentional case, thus the structure of Equations 1-3 is perfectly valid for us. Furthermore, a small distinction can be made in the convolution mechanism, depending on whether the node itself is taken into account or just the neighbors. In our work the node itself is always taken into account.

To simplify the GNN model, we specify a variation of (2) which is (4), where $\mathcal{N}(i)$ is the set of neighbors of node i.

$$\mathbf{y}_i^\ell = \mathbf{h}_\mathbf{i}^\ell + \sum_{j \in \mathcal{N}(i)} w_{ij} \mathbf{h}_j^\ell \tag{4}$$

The weights can either be predetermined or determined with attention, putting a GNN either in the convolutional or attentional category. In Graph Convolutional Neural Networks (GCN) [3] the weights are uniform, whereas in Graph Attention Networks (GAT) [19]. These are two of the most commonly used GNN architectures [4].

Other state-of-the-art GNN architectures are GraphSAGE [20], Simplified Graph Convolution (SGC) [21], and Graph Isomorphism Networks, (GIN) [22]. GraphSAGE provides an alternative convolution method where the node features are concatenated with the convolved features of its neighbors, instead of being part of the convolution themselves. SGC is an alternative simplified architecture that has been shown to perform much faster than other models without sacrificing prediction accuracy. Finally, GIN is a GNN that is designed to be as expressive as the Weisfehler-Lehman isomorphism test. All of these models have been shown to perform better on certain tasks but worse on others.

2.2 Hyperbolic geometry

Hyperbolic graph neural networks (HGNNs) are graph neural networks equipped with hyperbolic geometry, which is part of the fields differential geometry and Riemannian geometry. Furthermore, data can exhibit properties which are considered hyperbolic, making it suitable for HGNNs. In the following paragraphs, these concepts are explained in order to give a basis for understanding HGNNs.

2.2.1 Riemannian geometry

For an in-depth introduction into the mathematical foundation of differential geometry the reader is referred to ”Lectures on the Geometric Anatomy of Theoretical Physics” by Frederic Schuller [23]. Most concepts in that series are not required for this thesis and are omitted: from logic (propositions, logical operators, axiomatic systems) to set theory (Zermelo-Fraenkel axioms, equivalence, number spaces) to algebraic structures (groups, fields, algebras) to vector spaces (homeomorphisms, bases, tensors). All of these concepts build the foundation of differential geometry but are not necessarily required to work with it. In this review we present the minimum amount of theory that is needed to understand HGNNs. We start with topological spaces and later adopt the structure and notation used in Chami et al. [1].

A topological space is a set of points with a topology, where the topology defines neighborhoods (open sets) in the set. It can be seen as a generalization of metric spaces, where the closeness between points is defined, without assigning a numerical value, or metric, to it. A topology needs to satisfy three conditions which are omitted here.

A topological manifold is a topological space that is locally Euclidian, along with some other properties that we omit. Locally Euclidian means that each point has a neighborhood that is homeomorphic to an open subset of Euclidian space. The dimension of the Euclidian space defines the dimension of the respective manifold. A trivial example of a topological manifold is the Euclidian n-space $\mathbb{R}^n$, whereas a more interesting example is the torus, the Cartesian product of two circles, which locally resembles $\mathbb{R}^2$. Figure 3 shows how the torus can be embedded in 3-dimensional space, but it should be noted that this "ambient" higher-dimensional space is not necessary and the torus exists purely as a 2-dimensional manifold.

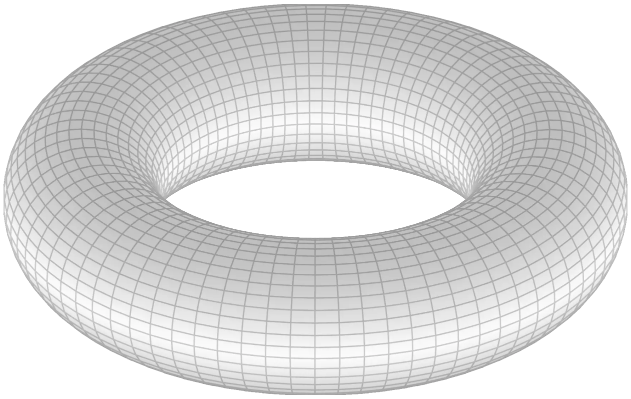


Figure 3: The torus, a 2-dimensional (smooth) topological manifold.

Furthermore, a manifold is defined by maps from the manifold to Euclidian space called charts. There are also transition maps that map from one charted set to another. In case these maps are infinitely differentiable, the manifold is called smooth. For example, the previously described torus is a smooth manifold. From now on we only consider smooth manifolds, the subject of the field of differential geometry, even when not explicitly mentioned.

A key feature of manifolds are tangent spaces, which are commonly used to perform operations in HGNNs. Consider a smooth manifold $\mathcal{M}$, and a point $\mathbf{x} \in \mathcal{M}$. The tangent space of $\mathcal{M}$ at $\mathbf{x}$ is denoted as $\mathcal{T}_{\mathbf{x}}\mathcal{M}$ and the union of all tangent spaces of $\mathcal{M}$ is called the tangent bundle, denoted by $\mathcal{TM}$. Tangent spaces can be defined intrinsically without considering a higher-dimensional ambient space in which the manifold is embedded, or extrinsically. Extrinsically means the tangent space is simply viewed as the tangent (hyper-)plane to the (hyper-)surface at a given point, or equivalently the first order approximation of the manifold around a given point. We adopt the extrinsic view as it fits our purpose and is easier to work with. Tangent spaces are very useful because of their vector space structure. An important property of the tangent space is that its dimension equals that of the manifold.

We further equip the manifold with a Riemannian metric. The Riemannian metric gives the manifold a measure the rate of change, and thus the distance between points, as well as a norm. A Riemannian manifold is thus a metric space. Using the Riemannian metric, Riemannian manifolds are also equipped with curve lengths and geodesics (generalization of shortest paths). Formal definitions of these can be found in Appendix C.

A crucial concept in differential geometry is that of curvature. Any formal definition of curvature is too involved for this summary but intuitively it describes how much a manifold deviates from Euclidian space. In Euclidian space, the curvature is 0 at any point. The curvature need not be constant over the whole manifold, though in this thesis we only work with manifolds with constant curvature. A manifold with constant positive curvature is called spherical, whereas a constant negative curvature means hyperbolic space. Many of the properties of Euclidian space do not hold for curved spaces and need to be defined properly. For more information on curvature in Riemannian manifolds we refer to Lee [24].

In Riemannian manifolds, points in the manifold can be mapped to any tangent space using the logarithmic map $\log_{\mathbf{x}} : \mathcal{M} \to \mathcal{T}_{\mathbf{x}}\mathcal{M}$, and vice versa using the exponential map $\exp_{\mathbf{x}} : \mathcal{T}_{\mathbf{x}}\mathcal{M} \to \mathcal{M}$, where $\mathbf{x} \in \mathcal{M}$. We refer to these maps by log-map and exp-map, respectively.

Another concept that needs to be generalized to non-Euclidian space is that of translation, generalized to parallel transport. The parallel transport $P_{\mathbf{x}\to\mathbf{y}} : \mathcal{T}_{\mathbf{x}}\mathcal{M} \to \mathcal{T}_{\mathbf{y}}\mathcal{M}$ transports a vector from the tangent space of one point to that of another, or, more generally, it preserves the local curvature when moving along a curve. In Riemannian manifolds parallel transport preserves the norm and the inner product with the direction of movement. In Euclidian space this is trivial but in curved space it has to be carefully defined and it can lead to unintuitive results.

For example, consider the globe as a spherical manifold. Suppose one is to stand on a point A on the equator facing the north pole N, then proceed to walk first to N, then to another point B on the equator (far away from A), then back to point A. This is to be done while only moving along geodesics (straight lines) and without changing one's orientation. Upon arrival back at A, one would find oneself facing a different direction than the original one towards N, as shown in Figure 4. In fact, the angle between the starting and final direction of view is proportional to the area of the triangle that the movement described. This phenomenon is called holonomy and is one of many examples where curved space behaves differently from Euclidian space.

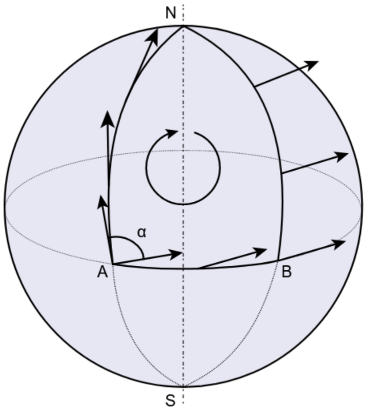


Figure 4: Visualization of parallel transport on a spherical 2-dimensional manifold. The tangent vector is transported from A to N to B to A, which results in some rotation.

2.2.2 Hyperbolic space

Having established the mathematical background of Riemannian geometry, we can now introduce hyperbolic geometry that is the basis for HGNNs. In short, hyperbolic space is a Riemannian manifold with constant negative curvature, along with being complete and simply connected, which is not something we will address here.

There exist different models that all describe the same hyperbolic space, two famous ones being the Poincaré ball model and the Hyperboloid model (also known as Minkowski model or Lorentz model) [8]. The two other commonly used models are the Poincaré half-plane model and the Beltrami-Klein model, though they are not used in this work. Both the Poincaré ball model and the Hyperboloid model have an intuitive interpretation with advantages and disadvantages. The d-dimensional hyperbolic space according to the Hyperboloid model is a hyperboloid surface embedded in $\mathbb{R}^{d+1}$, whereas the d-dimensional Poincaré ball is simply the unit ball in $\mathbb{R}^d$. As Riemannian manifolds, both also have a metric, which we give in Appendix C, along with other technical details of both models.

For the sake of visualization, we take $d = 2$, where we refer to the models as hyperboloid sheet and Poincaré disk respectively. Figure 5 shows the relationship between the two models. The disk has radius 1 and its center is at $(0, 0, 0)$, and the hyperboloid sheet has its apex at $(1, 0, 0)$, which is the origin of that hyperbolic space. The two models are related by a simple geometric operation in $\mathbb{R}^3$: a point on the hyperboloid sheet

can be projected onto the Poincaré disk by drawing a line between the point and $(-1, 0, 0)$. The intersection of this line and the disk is the projected point, and this transformation works both ways. It should be noted that the hyperboloid sheet stretches infinitely in all directions, whereas the Poincare disk contains the whole infitite space inside the unit disk (it gets denser and denser the closer one goes to the boundary).
The Poincaré ball model has an advantage when it comes to visualizing points in the space, in particular in the 2-dimensional Poincaré disk case. On the other hand, this model is less numerically stable than the Hyperboloid model [1], likely due to points close to the boundary being hard to distinguish with machine precision. This makes the Hyperboloid model the better choice for computational tasks. Furthermore, it is a better tool for visualizing the tangent spaces, as they are simply the tangent planes of the curved surface, in the 2-dimensional case.
For both models formulas are derived for distance, geodesics, tangent spaces, parallel transport, exponential and logarithmic maps, as well as projecting to the manifold from an outside point. Formulas for mapping between both manifolds and mapping between manifolds with different curvature also exist. All of these formulas are listed with explanation in Appendix C. It needs to be kept in mind that basic properties of Euclidian space often do not hold in hyperbolic space, meaning a lot of attention needs to be given to elementary operations.

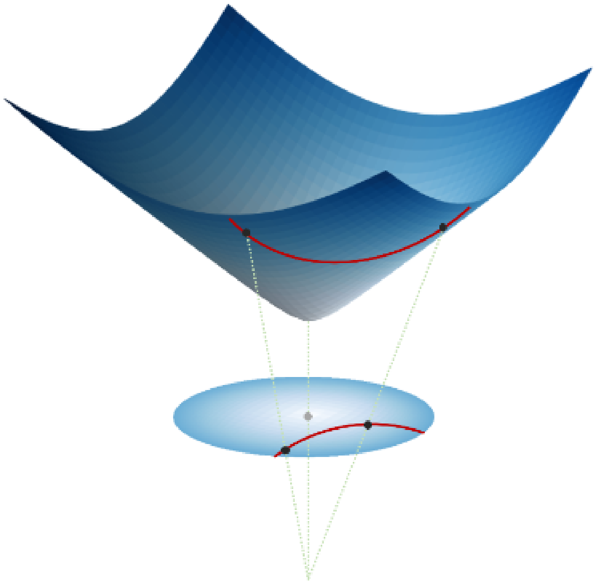


Figure 5: Two representations of 2-dimensional hyperbolic space, Poincaré ball (disk) and Hyperboloid model [1].

2.2.3 Hyperbolicity of graphs

Hyperbolic space has been a well-researched concept for a long time, but only relatively recently it has been used to embed data. This section contains an exploration of what it means for a graph to have underlying hyperbolic geometry.
There is a clear reason why hyperbolic space is preferred for embedding hierarchical or tree-like data: it expands similarly to tree-like data where Euclidian space does not [2]. It is well known that Euclidian space expands polynomially, meaning if we take a point in Euclidian space and consider a circle with radius r around it, the area A_E of the circle increases polynomially with r, specifically $A_E(r) = \pi r^2$. In contrast, hyperbolic space expands exponentially, with the area of a circle being $A_H(r) = 2\pi(\cosh r - 1)$ (this is for the case of curvature -1, a more negative curvature would mean faster increase).
We can quickly show a similar fact for graphs. Consider two graphs, a lattice/grid L and a tree T with a

constant branching factor, here we take the factor 3 (see Figure 6). Starting from a node in either graph, we can count the number of nodes $\hat{A}$ within a given distance $\hat{r}$. It can be shown that $\hat{A}_L(\hat{r}) = 2n^2 + 2n + 1$ and $\hat{A}_T(\hat{r}) = \frac{1}{2}(3^{n+1} - 1)$, for the lattice and tree respectively. The important thing here is that in the lattice the number of nodes within a given distance increases polynomially (in fact quadratically) with the distance, and exponentially in the hyperbolic case. Euclidian space simply does not have enough space to fit a tree. Based on this we can say that hyperbolic space is better for fitting tree-like and hierarchical data (which is also tree-like). We can further state can trees can be thought of as discrete hyperbolic spaces, or hyperbolic space as a continuous tree [25]. In a similar way, we can say that grids are Euclidian and cliques are spherical [26], though we will not go into these types of graphs.

The previous explanation gives a good intuition of why this is, whereas for a more technical approach we refer to [25]. For another visualization of this disparity, as well as a formal proof of the distortion in Euclidian space, we refer to [27].

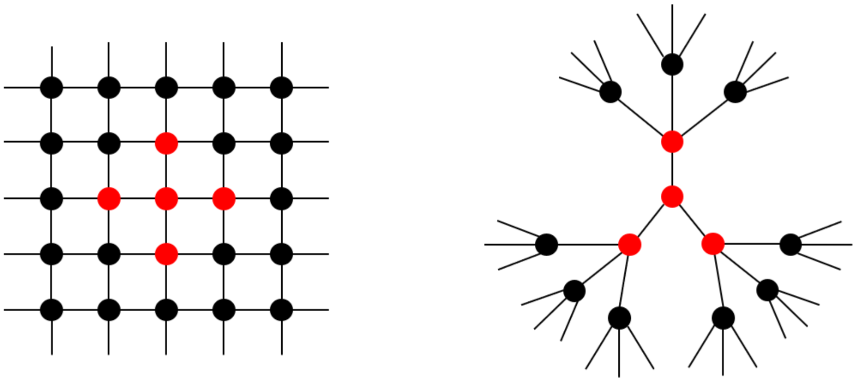


Figure 6: Expansion of a grid (left) and a tree (right). Nodes visited after 1 step are colored in red.

In short, trees are most suitably embedded in hyperbolic space. In fact it has been shown that a tree can be embedded in hyperbolic space with arbitrarily low embedding distortion even in low dimensions [28]. However, networks appearing in the real world are rarely exactly trees, which is why a measure of tree-likeness is useful. The most common one is Gromov's δ-hyperbolicity [29], which measures how tree-like a graph is by using its shortest paths. The δ-hyperbolicity is given by Definition 1 and is computed by iteratively sampling 4 points of the graph and calculating their δ value. This is necessary as calculating the value for all possible quadruples of points becomes infeasible for large graphs. Thus the maximum δ over some number of samples is usually taken and the graph is considered δ-hyperbolic.

Definition 1 *Let G be a graph with node set V, and let $\ell(x,y)$ be the shortest path distance between nodes x and y. G is δ-hyperbolic if for any nodes a, b, c, $d \in V$, ordered such that $\ell(a,b) + \ell(c,d) \geq \ell(a,c) + \ell(b,d) \geq \ell(a,d) + \ell(b,c)$, the following condition holds:*

$$(\ell(a,b) + \ell(c,d)) - (\ell(a,c) + \ell(b,d)) \leq 2\delta$$

δ-hyperbolicity is actually a measure for infinite graphs, measuring how the δ value increases as the graph increases. For trees, this value is constant, regardless of size, whereas for grids it increases with the size of the grid. It is this relationship between size and δ that determines hyperbolicity. However, for large graphs it is also a valid metric [29].

A 0-hyperbolic graph is exactly a tree, and in general low δ means the graph is close to a tree. It should be noted that cliques are also 0-hyperbolic but this and other examples will be discussed later.

Tree-likeness is not the only property associated with hyperbolicity of a graph. Common properties are heterogeneous/power-law/right-skewed degree distribution (PL), high/strong average clustering (HC), clustering inversely proportional to degree (IP), tree-like/close to a tree (TL), small-world/small diamater/short longest paths (SW). The correlation between various combinations of these properties has been studied [30]. Networks with PL are called scale-free. In scale-free graphs, low-degree nodes are called tail nodes, whereas high-degree nodes are called high-end nodes. Another class of graphs, hierarchical, is actually a description of real-world datasets, which have then been shown to exhibit certain properties: PL, HC, IP, TL [25]. Clearly, hierarchical graphs are scale-free, but the distinction comes from the fact that scale-free graphs may have clustering independent of node degree, which is not the case for hierarchical graphs. Furthermore, the concept "underlying hyperbolic geometry" refers to the same properties as hierarchical graphs. Thus we use hierarchical graphs and graphs with underlying hyperbolic geometry interchangeably. Both scale-free and hierarchical graphs have been shown to appear frequently in the real world and to exhibit high distortion in Euclidian embeddings [1], which is why it is important to consider them.

2.3 Hyperbolic graph neural networks

Having introduced the fundamentals of graph neural networks and hyperbolic geometry, we can continue with hyperbolic graph neural networks (HGNNs). We start with a history of the concept, listing major publications on hyperbolic learning in general but focusing on graph neural networks. Based on this search, we will summarize if and how each of our research questions has been addressed in literature.

A very early example of embedding data in hyperbolic space is Geographic Routing Using Hyperbolic Space [31], published in 2007, though the first exaple of graph embedding in hyperbolic space are Poincaré Embeddings [32], published in 2017. Poincaré embeddings are however only shallow embeddings and it should be noted that they are not the first examples of learning on Riemannian manifolds [33].

Various improvements have been made to shallow hyperbolic embeddings (for example introducing Lorentzian embeddings [34] which added the Hyperboloid/Lorentz model), but more importantly the first deep learning method that utilizes embedding in hyperbolic space is Hyperbolic Neural Networks, published in 2018 [9]. This work is considered pivotal for this field as it was the first to generalize neural network operations to hyperbolic space, specifically for MLPs and recurrent neural networks.

Based on HNN, hyperbolic geometry was applied into other deep learning architectures: convolutional neural networks [35], transformers [35], variational autoencoders [36], and graph neural networks [1]. Peng et al. [8] summarizes the major developments of the various branches of hyperbolic learning.

The first graph neural networks utilizing hyperbolic space are Hyperbolic Graph Neural Networks [10] and Hyperbolic Graph Convolutional Networks [1], both published in 2019. These were developed in parallel and have some differences in technical details and application area. Specifically, Hyperbolic Graph Neural Networks are for graph classification and HGCN is for link prediction and node classification. In short, these models work by considering each layer of the GNN as a hyperbolic space, as opposed to the Euclidian space used by most GNNs. In the modelling section we will go into the mechanisms of the HGCN but for now it is important to note that the operations in hyperbolic space that these models are built on are performed in tangent spaces.

Since then, these two models and the ones developed after them have been adapted to many specific types of networks, most notably knowledge graphs [37][38], recommender systems [13][39], and dynamic graphs [40][41].

Furthermore, some flaws were recognized in these first HGNN models, which have been addressed since. First, HGNNs were generalized to positive or negative curvature with κGCN, also called Constant Curvature GCN (2019) [27]. Furthermore, Chen et al. [11] proved that operations in the tangent space are only an approximation and do not lead to optimal results. This was addressed by models that perform their operations on the manifold: H2H-GCN [12], Lorentzian GCN [42], Fully-HNN [11] (all in 2021). Another limitation is that real-world data often does not have homogeneous curvature. This was addressed by GIL (Graph geometry Interaction Learning (2020) [43], where the embedding is done in Euclidian and hyperbolic space simultaneously. Similarly, mixed-curvature GNNs (2021) [44] embed data in a product of spaces of different curvatures. Most recently, κHGCN [45] (2022) attemps to include heterogeneous curvature in the models by calculating the Ricci curvature of each edge. Fu et al. [46] claims that curvature as hyperparameter (as in κGCN) or learnable parameter (as in HGCN) does not produce optimal results. This is fixed by performing adaptive curvature exploration using reinforcement learning in a new model called ACE-HGNN. Some attempts to go beyond Riemmanian manifolds have been made by Ultrahyperbolic NN (2021) [47] by embedding in pseudo-Riemannian manifolds, as well as trivial bundle embeddings (2021) [48]. Lastly, some papers develop completely new mechanisms for HGNNs, such as Hyperbolic Laplacian features [49], text enriched HGCN [50], adaptive frequency-based HGNN [51], MST-HGCN [52] (all in 2022).

The previous list is restricted to papers published before 2023. That being said, our literature research revealed that all papers on HGNNs released in the first half of 2023 fall within one of these categories. Thus this is an exhaustive list of general directions for new HGNN models.

Yang et al. [2] is a great resource summarizing most HGNN architectures, down to fine differences in mechanisms. They also list persisting problems with them. In short these problems are the presence of heterogeneous geometry of real-world data, the lack of purely hyperbolic optimization methods, the lack of understanding of why and when HGNNs are beneficial, and lack of scalability of hyperbolic models. Most of the papers listed previously attempt to fix one or more of these problems, though they are largely still open.

Next we investigate in how far our research questions have been addressed in the previously mentioned literature. First, we look at measures of hyperbolicity that are used to predict or explain model performance. Some papers only use datasets with known high hyperbolicity, where something is already known about the data (for example trees or pure hierarchies [10]), whereas others distinguish their datasets based on some measure. There does not seem to be a consensus on the best measure to use, though δ-hyperbolicity appears to be the most common by far. It is used by the majority of HGNN papers, including the HGCN [1] and papers that build on it.

As a variation of this, the average δ, called δ_{avg} can be used, where instead of taking the largest found value, the average over all samples is taken. Zhang et al. [42] uses this metric, as well as Tifrea et al. [53], the latter of which scales this value based on the size of the graph. Furthermore, Zhu et al. [43] looks at the distribution of the δ-curvature (of largest connected element without sampling). The motivation behind using these alternative measures is to surmount the limitations of standard δ-hyperbolicity, which will be addressed later in this thesis.

Traditional graph features associated with hyperbolicity are also often looked at, such as network size, degree distribution (often the power of the power-law of the degree distribution) [54][55], average degree [56], clustering coefficient [54]. Density is often considered when working with recommender systems, though sometimes also for more general application [50]. However, it needs to be kept in mind that density does not have any theoretical connection to hyperbolic geometry.

In some papers, completely different measures are introduced and used. A measure of sectional curvature/parallelogram rule [38][27]. On the other hand, assortativity is a measure used in trivial bundle embeddings [48], though this is not actually an HGNN.

With some slight variations, these are all the measures used in HGNN literature to determine hyperbolicity and thus predict the performance of the respective HGNN. In general, most research agrees that hyperbolic models are better for data with underlying hyperbolic geometry, though some claim that Euclidian models are sometimes more useful [57]. Furthermore, there does not seem to be serious study on the robustness to noise of HGNNs.

As far as the second research question goes, concerning curvature as a learnable parameter, we do not have as many sources to look into. A lot of HGNN architectures simply set the curvature as a hyperparameter [10]. Chami et al. [1] prove that the layers of the GNN are equally expressive regardless of curvature, but changing curvature can make the whole model more numerically stable. In fact, they show that the accuracy of the model is directly influenced by the value of the layer curvature. One of the later works, ACE-HGNN [46], shows that this way of setting the curvature as a learnable parameter is not optimal, instead opting to use reinforcement learning to achieve the same thing, leading to better results. This is however a completely different mechanism and does not help answer our research question. Thus the only known results about learnable curvature come from Chami et al.

Concerning the effects of the embedding dimension on model performance, the majority of research agrees that HGNNs are most beneficial for low-dimensional embedding [10]. This means that at low embedding dimension, HGNNs perform significantly better than Euclidian GNNs, whereas at high embedding dimension the two types of models are comparable in performance. This trend has been identified when the dataset is hyperbolic, but it appears that it has not been investigated what the results looks like when the data is not hyperbolic.

3 Modelling

This section starts with an in-depth view of the HGCN architecture. Then we will explain the modelling choices like benchmark models, datasets, and hyperparameters. Finally, three random graph models are introduced, as well as dataset statistics related to hyperbolicity.

3.1 HGCN

The Hyperbolic Graph Convolutional Network (HGCN) [1] is an HGNN model, which introduces hyperbolic geometry to a GNN. There are several challenges addressed by the HGCN: How to use Euclidian features for hyperbolic operations? How to do the aggregation of the features in hyperbolic space? How to choose the curvature for the hyperbolic space (for each layer)? All of these questions are trivial (or non-existent) for regular/Euclidian GNNs but it is not obvious how to translate them to the hyperbolic case. More concretely, the HGCN needs to redefine the core operations of a graph neural network (1), (4), and (3) (specifically those of the GCN that it is based on), to satisfy these goals .
In this section we introduce the superscript ^{E} to signify the respective point lies in Euclidian space. For hyperbolic space we will use the superscript ^{H}, and, where needed, the curvature $-1/K$ will be indicated with a superscript ^{K}, with $K > 0$.
Before defining the message passing operations, features must be mapped hyperbolic space, as they are usually Euclidian. This is done by considering them as vectors in the origin tangent space, and using the exponential map to map them to hyperbolic space. The formula for the exponential map can be found in Appendix C.
In Euclidian GNNs the feature transformation step is a simple matrix multiplication and vector addition (1). However, this vector structure does not exist on the hyperbolic manifold, thus the authors opted to use the tangent space for its vector structure. Other options are possible including performing the operations directly on the manifold, which should lead to more precise results at the cost of computational power.
Thus the necessary operations are redefined as (5) and (6), where W and $\mathbf{b}$ again simply represent the weights and bias of the layer, respectively. Clearly, both operations are dependent on the manifold curvature of the layer. Here the superscript $^\ell$ is omitted but it is implied that all the embeddings here are in the same layer.

$$W \otimes^K \mathbf{x}^H := \exp_{\mathbf{o}}^K \left(W \log_{\mathbf{o}}^K \left(\mathbf{x}^H \right) \right) \tag{5}$$

$$\mathbf{x}^H \oplus^K \mathbf{b} := \exp_{\mathbf{x^H}}^K \left(P_{\mathbf{o} \to \mathbf{x}^H}^K (\mathbf{b}) \right) \tag{6}$$

In (5) we log-map down to the tangent space of the origin, where we perform the multiplication, followed by mapping back up to the manifold with the exp-map. In (6) we consider $\mathbf{b}$ as a vector in the origin tangent space, which we parallel transport to the tangent space of the point we are trying to add it to. Then we exp-map it up to the tangent space.
The HGCN is equipped with different options on how to perform the neighborhood aggregation. The most basic method is convolution, which can be done either in the local tangent space of the point, as in (7), or in the origin tangent space. As in a regular GCN, the weights w_{ij} of the averaging are uniform. Here as well layer indication is omitted. In (7), as opposed to the Euclidian case (4), the node itself disappears in the convolution, as log-mapping it to its own tangent spaces simply gives 0.

$$\mathrm{AGG}^K \left(\mathbf{x}^H \right)_i = \exp_{\mathbf{x}_i^H}^K \left(\sum_{j \in \mathcal{N}(i)} w_{ij} \log_{\mathbf{x}_i^H}^K \left(\mathbf{x}_j^H \right) \right) \tag{7}$$

The other option is to use attention, meaning the weights are learned as in (8). This simply means that to find the weight w_{ij} to be used for the respective aggregation, the embeddings of each point in the current layer are log-mapped to the origin tangent space and concatenated. This concatenation is then fed to a separate multi-layer perceptron, followed by a softmax over all neighbors.

$$w_{ij} = \mathrm{SOFTMAX}_{j\in\mathcal{N}(i)}\left(\mathrm{MLP}\left(\log_{o}^{K}\left(\mathbf{x}_i^H\right) \| \log_{\mathbf{o}}^{K}\left(\mathbf{x}_j^H\right)\right)\right) \tag{8}$$

In summary, there are two choices to be made in aggregation: basic convolution or attention; aggregation on local tangent space or origin tangent space.

The HGCN must also define an activation function. Aside from the usual purpose of the nonlinear activation function, in the HGCN it has another goal: to transform the curvature between the layers. In the HGCN each layer can have its own curvature, thus the embeddings need to be transformed between layers.

Given hyperbolic curvatures $-1/K_{\ell-1}, -1/K_\ell$ at layer $\ell-1$ and ℓ respectively, we will use $\sigma^{\otimes^{K_{\ell-1},K_\ell}}$ to denote the activation between them (9).

$$\sigma^{\otimes^{K_{\ell-1},K_\ell}}\left(\mathbf{x}^H\right) = \exp_{\mathbf{o}}^{K_\ell}\left(\sigma\left(\log_{\mathbf{o}}^{K_{\ell-1}}\left(\mathbf{x}^H\right)\right)\right) \tag{9}$$

This process consists of taking a point and log-mapping it down to the origin tangent space $\mathcal{T}_{\mathbf{o}}\mathbb{H}^{d,K_{\ell-1}}$, applying the Euclidean non-linear activation σ, and then exp-mapping it up to $\mathbb{H}^{d,K_\ell}$. This is possible because the origin tangent spaces for hyperbolic spaces with different curvature are the same, so it can be used as a bridge between the layers.

The layer curvature mentioned previously can be set to a desired value, which can affect model performance. However, it can also be set as a learnable parameter in the neural network. This has been shown to lead to better results in most cases, even though the expressive power can be shown to be the same regardless of curvature. This is on the one hand due to machine precision, and on the other hand because the norms of hidden layers vary according to curvature [1]. This means that if a bad value is chosen for the curvature, despite the same expressive power, the optimal weights of the layer may be very large or very small. Thus, it can lead to computational issues.

For the node classification task, the output of the last HGCN layer is mapped to the tangent space of the origin with the logarithmic map. There, Euclidean multinomial logistic regression can be done. An alternative approach would be to directly classify points on the hyperboloid manifold using the hyperbolic multinomial logistic loss. This method has been shown to perform similarly to the previous one, so the choice is made to not use it [1].

For link prediction, the Fermi-Dirac decoder is used (10), a generalization of the sigmoid to hyperbolic space [25].

$$p\left((i,j)\in\mathcal{E} \mid \mathbf{x}_i^{L,H}, \mathbf{x}_j^{L,H}\right) = \left[e^{\left(d_{\mathcal{L}}^{K_L}\left(\mathbf{x}_i^{L,H},\mathbf{x}_j^{L,H}\right)^2 - r\right)/t} + 1\right]^{-1} \tag{10}$$

The result is the probability of a link between $\mathbf{x}_i$ and $\mathbf{x}_j$, with r and t being tunable parameters. The neural network is trained to optimize using cross-entropy loss using negative sampling, meaning the negative class (pairs of node without an edge between them) is undersampled in order to match the positive class.

Summary

Putting everything together, we end up with the following sequence of operations taking place in the HGCN. First, points are mapped to hyperbolic space. Then, a number L of hidden layers follows, each with 3 core operations (11), (12), and (13).

$$\mathbf{h}_i^{\ell,H} = \left(W^\ell \otimes^{K_{\ell-1}} \mathbf{x}_i^{\ell-1,H}\right) \oplus^{K_{\ell-1}} \mathbf{b}^\ell \tag{11}$$

$$\mathbf{y}_i^{\ell,H} = \exp_{\mathbf{x}_i^H}^{K}\left(\sum_{j\in\mathcal{N}(i)} w_{ij}\log_{\mathbf{x}_i^H}^{K}\left(\mathbf{x}_j^H\right)\right) \tag{12}$$

$$\mathbf{x}_i^{\ell,H} = \sigma^{\otimes^{K_{\ell-1},K_\ell}}\left(\mathbf{y}_i^{\ell,H}\right) \tag{13}$$

Finally, the embeddings of the final layer $(\mathbf{x}^{L,H})_{i\in\mathcal{V}}$ are used for node classification by mapping down to the origin tangent space and performing Euclidian classification, or link prediction by using the Fermi-Dirac

decoder.
In practice, an extra step of projecting the new point onto the manifold or onto the tangent space is done to ensure numerical stability. However, this is not necessary when using the Poincaré ball model.

3.2 Modelling choices

There are several choices to be made for the modelling of this thesis. As a foundation we use the code provided by Chami et al. [1] as it provides a vast and proven library of models, datasets, tasks, and hyperparameters.

Architecture

As benchmark models to the HGCN, Chami et al. use a selection of shallow, deep, Euclidian, hyperbolic, MLP, and graph models. The shallow models include Euclidian graph embedding, as well as its hyperbolic counterpart, Poincaré embedding [32]. As shallow methods, these cannot incorporate node features, thus so-called "mixed" versions of those are also used, where node features are simply concatenated to the shallow embedding.
For standard models the options are a multi-layer perceptron, as well as its hyperbolic counterpart, the HNN [9]. These methods make use of node features but not of the graph structure.
Lastly, state-of-the-art GNN models are provided, the main limitation of which is not using hyperbolic space. These are the previously mentioned GCN, GAT, GraphSAGE, and SGC.
We choose the GCN as a comparison model. We want the two models to be compared to have as similar structure as possible, differing only in the use of hyperbolic space. An alternative approach would be to use the HGCN with attention and the GAT. However, the attention mechanism is not the focus of this work, and it significantly increases the training time for experiments, so we opt to use the HGCN without attention, and the GCN. Other methods like GraphSAGE and SGC that differ in technical details are not interesting when studying the effects of hyperbolic geometry.

Datasets and task

The HGCN has been benchmarked on 5 datasets, which are also often used to benchmark GNN architectures. We will use all the ones that are provided in the implementation, meaning with the notable absence of the Human PPI dataset from the original paper, as well as a different version of the Disease dataset, called Disease-LP (for link prediction) [1]. A detailed overview of these datasets is given in Table 1. As task we take link prediction, as it is applicable even when the nodes are not classified.
Furthermore, we will use a a dataset called Road, a road network, as an example with clear non-hyperbolic geometry. We know that no hyperbolic geometry is present as the graph is essentially already embedded in $\mathbb{R}^2$. To extract this dataset we use OpenStreetMaps, which is commonly used for this purpose [58]. We extract the road network of Enschede, Netherlands by taking a square around the city center with side 10km, resulting in a graph with comparable size to the others (Figure 7). In this dataset the node features are simply the geographic coordinates of each node (road intersection), and we ignore the provided edge features (number of lanes, speed limit, etc). We do not introduce node classes since the only task will be link prediction.

Hyperparameters

There are many hyperparameters to be chosen for the experiments. The implementation of Chami et al. [1] provides their found hyperparameters for some datasets and models, most importantly for the HGCN. We use those as a starting point and make changes where necessary.
First we discuss the geometry-related hyperparameters. As a hyperbolic model, the Poincaré ball will be used, and the aggregation will done in the origin tangent space as opposed to the local tangent space. The attention mechanism will not be used as to make the HGCN directly comparable to the GCN. Unless stated otherwise, the curvature of each layer will be set as a learnable parameter. For the Fermi-Dirac decoder parameters we always use the default $r = 2$ and $t = 1$ [1].

Dataset	Description	δ
Disease	Simulated disease spreading	0
Airport	Airline routes	1
Pubmed	Citation network for medicine publications	3.5
Cora	Citation network for machine learning publications	11
Road	Road network of Enschede	20

Dataset	Nodes		Edges	
Disease	2665	Agent	2664	Infection
Airport	3188	Airport	18631	Route exists
Pubmed	19717	Publication	88651	Citation
Cora	2708	Publication	5429	Citation
Road	5160	Road intersection or endpoint	7009	Road

Dataset	Features		Classes	
Disease	11	Susceptibility to disease	2	Infected or not
Airport	4	Long., lat., alt., GDP of country	4	Country pop. (4 bins)
Pubmed	500	Presence of dictionary words	3	Academic subarea
Cora	1433	Presence of dictionary words	7	Academic subarea
Road	2	Coordinates	-	-

Table 1: Datasets to be used in experiments.

As far as the neural network hyperparameters, the following have to be chosen: learning rate, weight decay, dropout, number of hidden layers, and activation function. Chami et al. report performing a hyperparameter search over all of these, so we use their results as a starting point. We simplify the search by picking ReLU as the activation function, and setting the number of hidden layers to 2, to make all experiments comparable to each other and to the experiments of Chami et al. (there the number of layers is also always chosen to be 2, with 16 neurons each, presumably for the same reason). Normalization of the features is done before training.

Dropout is done in the form of DropConnect, which is a generalization of dropout [59]. In dropout, nodes in the neural network are randomly ignored to prevent overfitting. In DropConnect, actual connections are dropped instead of nodes. This is necessary since dropout itself cannot be directly used on the HGCN due to the curvature in the hidden layers, whereas the weights themselves are in Euclidian space.

The optimizer used is Adam. Note that a special optimizer like RiemannianSGD is not needed for the HGCN since the neural network parameters are all in Euclidian space. The method is batch gradient descent, meaning the whole dataset is used in every optimizer step.

Having chosen the number and size of hidden layers, as well as the activation function, this leaves learning rate, weight decay, and dropout to be chosen. For learning rate we consider the values 0.001 (low), 0.01 (medium), and 0.1 (high); for dropout we consider 0.0 (none), 0.2 (low), and 0.5 (high): for weight decay we consider 0.0 (none), 0.0001 (low), and 0.001 (high).

We use the following procedure to find optimal hyperparameters. The respective model is trained and evaluated on the validation set for 5 runs with different initial (random) weights. We try to achieve a) high average validation accuracy, b) consistency between runs (in both accuracy and epochs before early stopping), and c) smoothly decreasing loss. Observing all three of these should ensure that the chosen hyperparameters are suitable.

First, the best learning rate is picked without any dropout or weight decay. After this, with the set learning rate, the best dropout value is found. Finally, with the found learning rate and dropout values, the best weight decay value is found. There are more involved methods of finding optimal hyperparameters, but this suffices for this thesis.

Furthermore, we use a data split of 85/5/10% for training/validation testing. This split is not changed between runs. The evaluation is done using the ROC AUC, the area under the ROC curve. A value of 1

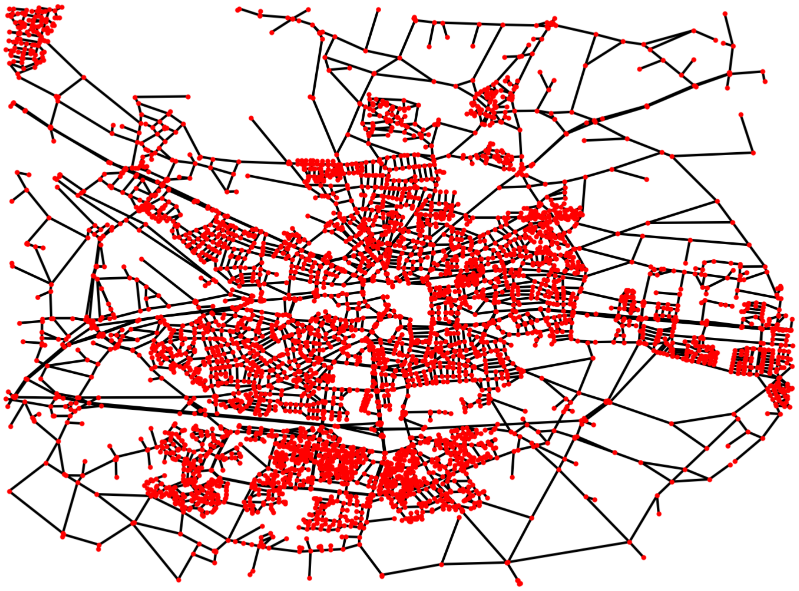


Figure 7: Road dataset. Each red point is a road insterstection, and is plotted in accordance with its given geographic coordinates.

indicates a perfect prediction, 0 a completely wrong prediction. In this work "accuracy" is sometimes used interchangably with ROC AUC, and the ROC AUC may be given scaled as a percentage (i.e. 90.0 instead of 0.9).

The number of training epochs is 5000 but early stopping is done based on the validation set with patience of 100 epochs. This means that if the ROC AUC on the validation set does not improve in 100 consecutive epochs, early stopping occurs. In practice early stopping occurs almost always, so we can also take the number of epochs before early stopping into consideration. The patience may be adjusted if necessary, though this should be avoided as it might make the results less comparable to each other. Lastly, we may make use of a learning rate scheduler, though the Adam optimizer already adjusts the learning rate of each parameter so this might not be necessary.

Appendix A contains the found optimal hyperparameters for each dataset. We omit the specifics of finding these, though one observation should be made. Regardless of hyperparameters used, training the GCN on the HRG dataset always has a large variability between runs. This shows an element of randomness in this model, specifically on this dataset, which can likely be attributed to the model space not being suitable (Euclidian space for hyperbolic data).

3.3 Random graph models

We introduce three random graph for which something is already known about the underlying geometry. In experiments we use same random seed for generating these, effectively making each a unique dataset. The stochastic block model (SBM) [60] creates blocks of nodes of given sizes. Every pair of nodes connects independently with some given in-block or between-block connection probability. We opt to use high within-block and low between-block connection probabilities, resulting in close communities and a dense graph, as shown in Figure 8. For experiments we will use the same parameters as Figure 8 with the block sizes multiplied by 10. Regardless of any properties we might find with experiments, the SBM should not have any underying hyperbolic geometry. It is not tree-like and the degree distribution is very homogeneous, though there is some clustering present.

Another model with a similar structure is the Lancichinetti–Fortunato–Radicchi benchmark (LFR) [61].

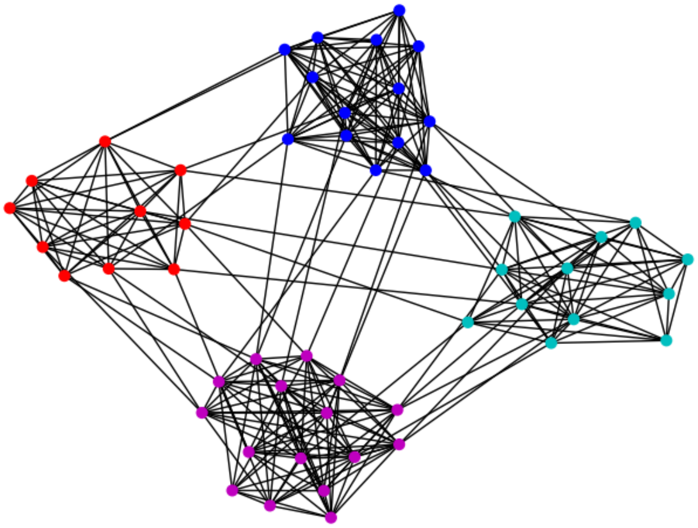


Figure 8: Example of graph generated by the stochastic block model. Block sizes are 10, 12, 14, and 16. Within-block connection probabilities are between 0.8 and 0.9. Between-block connection probabilities are between 0.02 and 0.1.

It exhibits the same community structure as the SBM but it ensures heterogeneous degree distributions, making it more comparable to real-world hierarchical datasets. It is created by specifying the power of the degree distribution τ_1, the power of the community size distribution τ_2, the intra-community connection probability μ, as well as some parameters constraining the degrees. We will use the same parameters as in Figure 9 with 1000 nodes. This dataset is similar to the SBM in structure, with the notable addition of heterogeneous degree and block distributions, making it closer to a scale-free network.
Lastly, we consider hyperbolic random graphs (HRG) [62]. These are generated directly based the on principle of underlying hyperbolic geometry [25]. In short, points are produced on the unit disk with uniform angular probability and exponential radial probability (high probability of being anywhere close to the edge of the disk, low probability of being close to the center). Then, points are connected which are close to each other in terms of hyperbolic distance. Aside from exhibiting a power-law degree distribution, high clustering, and small diameter, HRGs come with hyperbolic coordinates for each node, laying on the Poincaré disk.
This graph model also comes with tunable parameters. α controls the power of the degree distribution and thus the magnitude of the underlying curvature. The power is calculated by $2\alpha + 1$, with α ranging from 0.5 to 1. T is the temperature controlling the edge generation. With temperature 0, only nodes that lie close to each other in hyperbolic space will be connected. As T increases, these connections become more and more random. The HRG is a very hyperbolic graph by many properties and it is expected that it would be best embedded in hyperbolic space. Perhaps the same could be said for the LFR considering its predefined heterogeneous degree distribution, but for SBM the expectation is the opposite. In any case, it needs to be checked what statistics can be extracted from these graph models and how the two GNN models perform on them. There are of course many other random graph models with varying underlying geometry, but we will focus on these three as they represent the extremes (purely hyperbolic to non-hyperbolic).

3.4 Data statistics

Here we provide graph properties that can be measured against model performance. We base this selection on existing literature on the topic. This shows the majority of research that compares hyperbolicity and model performance use either basic measures, δ-hyperbolicity-based measures, or in some rare cases entirely different measures. We choose to look at the basic features and δ-hyperbolicity. We do not include less

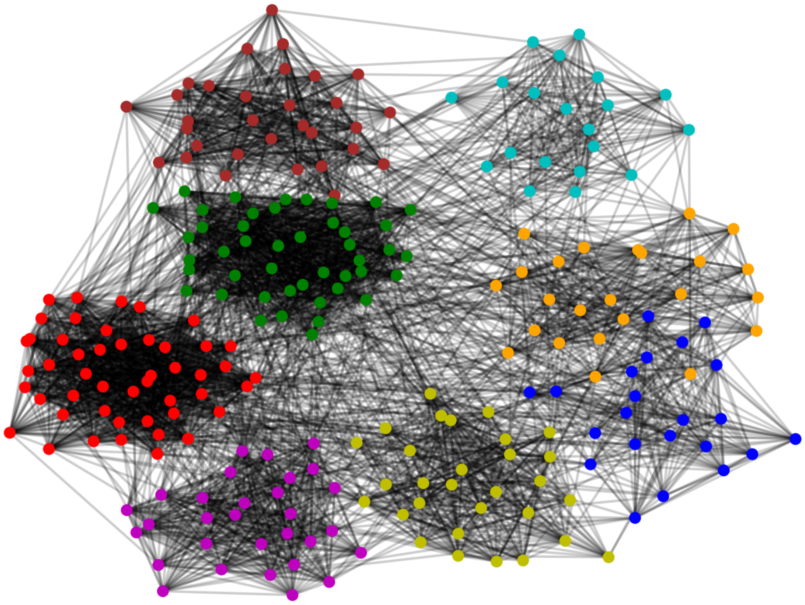


Figure 9: LFR benchmark graph with 250 nodes and parameters $\tau_1 = 2$, $\tau_2 = 1.1$, $\mu = 0.1$, minimum degree 20, maximum degree 50. The nodes of each block are pictured with a different color.

common properties like sectional curvature. Instead we add a novel method in the form of Balanced Forman curvature of the edges of a graph, which has previously been used to detect bottlenecks in graphs [26].

3.4.1 Basic measures

Based on theory related to hierarchical/scale-free networks and underlying hyperbolic geometry, we can investigate some of the traditional graph features. Specifically: network size, density, average degree, degree distribution (to identify power-law/heterogeneous distribution), average clustering coefficient, clustering coefficient distribution, clustering-degree relationship.

3.4.2 Gromov's δ-hyperbolicity

Gromov's δ-hyperbolicity measures how tree-like a graph is in terms of its metric structure [29] and is a separate measure than traditional properties like clustering and degree distribution [30]. The lower δ is, the more tree-like a graph is, with trees being 0-hyperbolic. Definition 1 gives the formal definition of δ-hyperbolicity.

The δ-hyperbolicity of basic graph types can easily be computed. For trees and complete graphs/cliques it is 0, where as for a grid of size n-by-n it is $n-1$ [29]. It is noteworthy that complete graphs are 0-hyperbolic even though in many ways they are quite different from trees. More concretely, 0-hyperbolic graphs are exactly clique trees, that is cliques connected in a tree shape. Furthermore, for cycles it is approximately $n/4$, which is the largest possible hyperbolicity for a graph of size n.

δ-hyperbolicity is a measure for infinite graphs but it is still a valid way of comparison if graphs are sufficiently large. For example, the δ of a 3-by-3-node grid is 2, implying high hyperbolicity. However, a 20-by-20-node grid would be 19. The important thing to note here is that for a graph with no hyperbolic geometry, δ increases with the size of the graph. In contrast, for graphs with hyperbolic geometry δ is independent of the size. Thus, if computed for reasonably large graphs, δ should give a good indication of hyperbolicity.

That being said, we will use δ-hyperbolicity as a metric for how hyperbolic a graph is. As the graphs we are

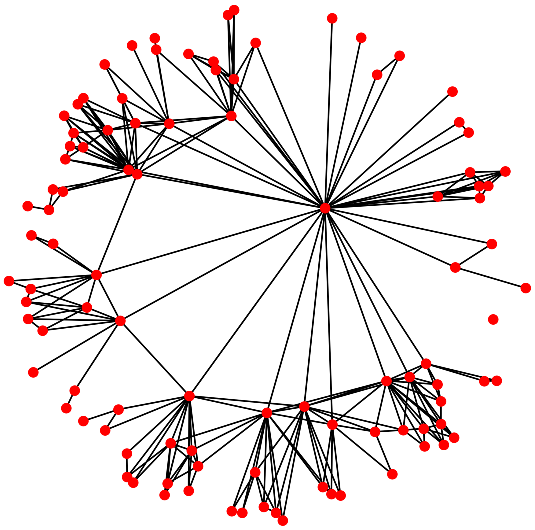


Figure 10: Hyperbolic random graph with 100 nodes, $\alpha = 0.75$ (default), $T = 0$.

working with are quite large, we will use sampling with 50000 samples. According to the definition, we take the largest found δ of all samples. Additionally, based on what other researchers have introduced, we will also look at δ_{avg}, which is the average value over the samples. Other known options, like the distribution of δ over the found samples, will not be used.

3.4.3 Balanced Forman curvature

To provide a completely different point of view on graph geometry, we add a measure of curvature of an edge based on Ricci curvature. There is no single measure of edge curvature so we take Balanced Forman curvature, a recent addition which was developed to detect bottlenecks in graphs [26].
Yang et al. [2] proposes to use such a curvature measure for models that take into account heterogeneous curvature, but here we use it as a measure for global homogeneous curvature. We can aggregate in two ways to provide information about the whole graph: taking the average over all edges, and looking at the overall distribution.
Balanced Forman curvature of the edge between two nodes i and j is given by Definition 2. Here d_u is the degree of node u, $\sharp_{\triangle}(u,v)$ is the set of triangles containing edge (u,v), $\sharp_{\square}^{u}$ is the set of neighbors of u forming a 4-cycle based at (u,v) without diagonals. $\gamma_{\max}(u,v)$ is the maximal number of 4-cycles based at (u,v) traversing a common node.

Definition 2 *Given two adjacent nodes i and j in a graph, the Balanced Forman curvature of the edge between i and j is given by*

$$\mathrm{Ric}(i,j) := \frac{2}{d_i} + \frac{2}{d_j} - 2 + 2\frac{|\sharp_{\triangle}(i,j)|}{\max\{d_i,d_j\}} + \frac{|\sharp_{\triangle}(i,j)|}{\min\{d_i,d_j\}} + \frac{(\gamma_{\max}(i,j))^{-1}}{\max\{d_i,d_j\}}\left(|\sharp_{\square}^{i}| + \left|\sharp_{\square}^{j}\right|\right).$$

An important observation is that the values of $\mathrm{Ric}(i,j)$ range from -2 (hyperbolic) to infinity. We can look at a few basic graph types to investigate their curvature.
Example: Let C_n be a cycle on n nodes. Due to symmetry all edges will have the same curvature. For the cases $n = 3$ and $n = 4$ the curvature is $\frac{3}{2}$ and 1, respectively. For the case $n \geq 5$ the curvature is 0.
Example: Let K_n be a complete graph on n nodes. The curvature of each edge is $\frac{n}{n-1}$.
Example: Let G_n be a grid of size n-by-n. This graph is not fully symmetric so internal and boundary edges will have a different curvature. Interior edges have curvature 0, edges on the exterior have curvature $\frac{1}{3}$ or $\frac{1}{6}$.
Example: Let T_r be a tree with a constant branching factor r. We again have to distinguish different edges,

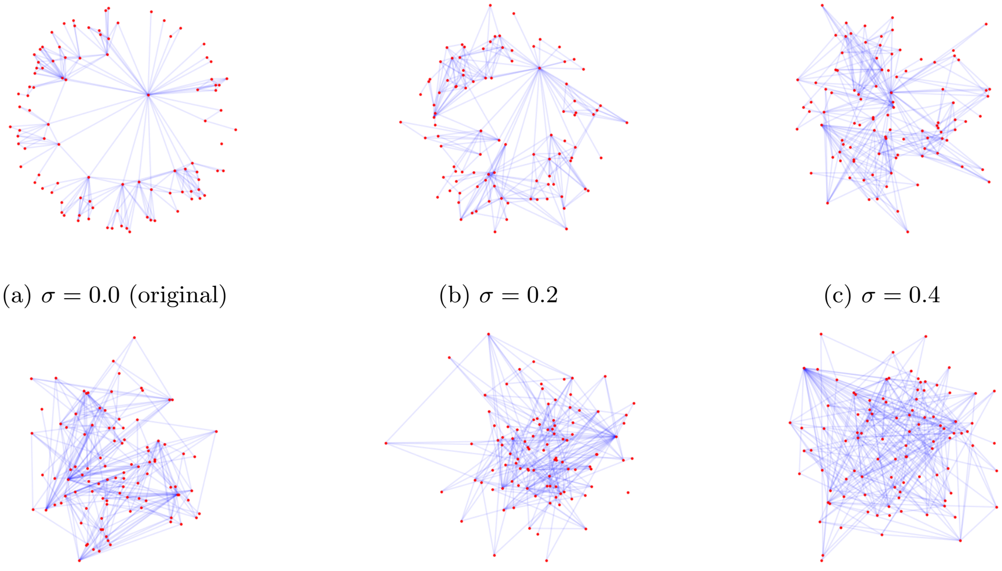
this time leaf and non-leaf edges (an edge is a leaf edge if it is connected to a leaf node). Leaf edges have curvature $\frac{2}{r+1}$, non-leaf edges have curvature $\frac{4}{r+1} - 2$. Suppose $r = 3$, then the leaf edges have curvature $\frac{1}{2}$ and the rest of the edges have curvature -2.

We have seen that, when looking at basic graphs, Balanced Forman curvature provides different types of values than δ-hyperbolicity. Most notably cliques have positive Balanced Forman curvature. On the other hand, the Balanced Forman curvature for grids is (close to) 0, further differentiating this measure from δ-hyperbolicity. Trees have negative curvature which matches other notions of graph curvature.

3.4.4 Noise

We will analyze the robustness to noise of the HGCN and GCN on the HRG data. The reason HRG is useful for this is because it is easy to add noise to it in a structured way, and we know it has underlying hyperbolic geometry. We will do this in two ways: introducing noise to the graph structure in the form of changing the temperature parameter T in the HRG generation algorithm; and introducing noise to the features by adding values sampled by a normal distribution with mean 0 and standard deviation σ.

Figure 11 and Figure 12 show the result of the noise on the data. Setting the temperature parameter T close to 1 results in very few edges which is not desirable for experiments, which is why we do not take values above 0.8. To achieve fully random features, we simply replace them by values sampled by a standard normal distribution. For fully random egdes, we generate edges from scratch, each with a probability 0.05, as in the Erdős–Rényi model [63]. Furthermore, we can utilize both types of noise in the same experiments, with varying parameters.



(a) $\sigma = 0.0$ (original)

(b) $\sigma = 0.2$

(c) $\sigma = 0.4$

(d) $\sigma = 0.6$

(e) $\sigma = 0.8$

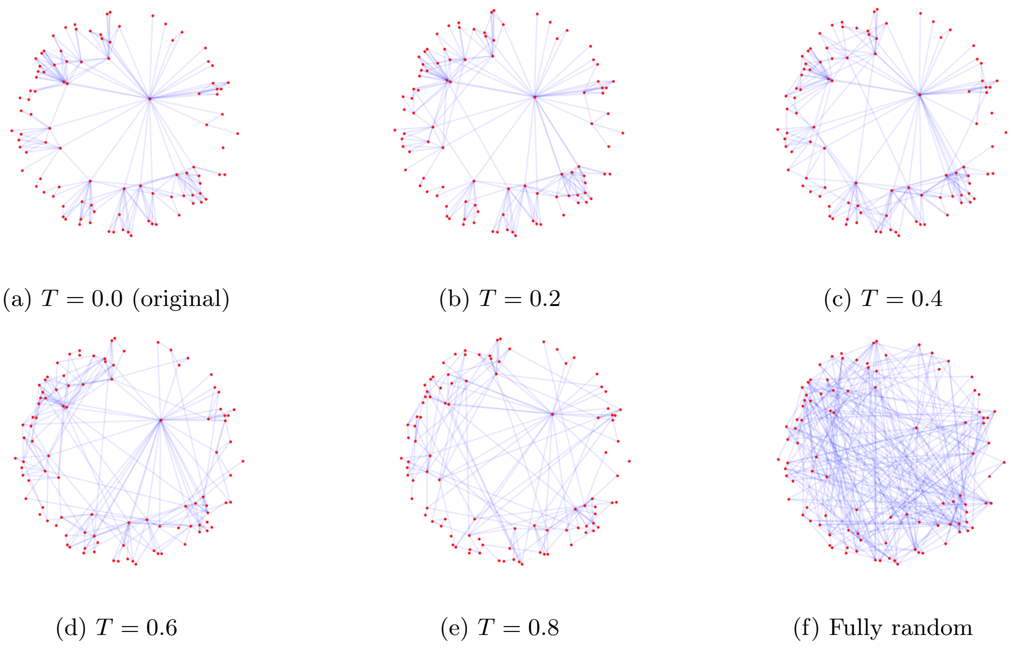
(f) Fully random

Figure 11: Adding noise to features of HRG.

4 Experiments

In this section we present experiments that attempt to answer the research questions of this thesis. The first subsection presents a breakdown of the selected properties of each dataset, comparing it to the performance of both the GCN and HGCN. This is followed by an analysis of the effect of noise on model performance, specifically on the HRG dataset.

In the second subsection, the role of the learnable curvature in the HGCN is investigated. Finally, the third subsection addresses the performance gap between GCN and HGCN by varying the embedding dimension.


(a) $T = 0.0$ (original)

(b) $T = 0.2$

(c) $T = 0.4$

(d) $T = 0.6$

(e) $T = 0.8$

(f) Fully random

Figure 12: Varying temperature of HRG.

4.1 Dataset properties and model performance

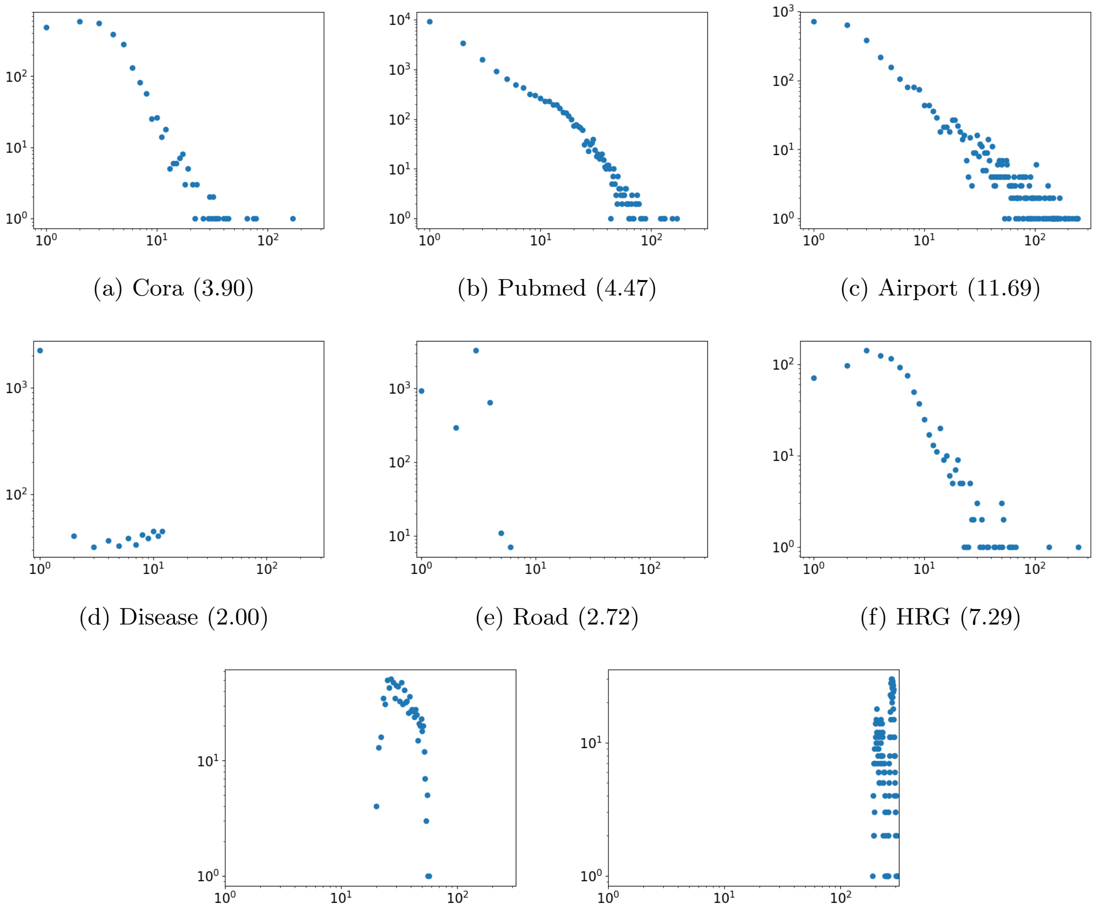
We will start by investigating the first reserach question, namely what dataset properties relate to model performance in the case of GCN and HGCN.

4.1.1 Dataset statistics

First, we observe the size and density of each dataset in Table 2. With the exception of Pubmed, the real-world datasets are of comparable size, with Pubmed being much larger. The synthetic datasets based on graph models (HRG, LFR, SBM) are all chosen to be (around) 1000 nodes in size. The density of the real-world datasets is also similar, Cora and Airport being about one order of magnitude larger. The graph model datasets vary quite a bit in density, due to the chosen parameters for LFR and SMB. The SBM's large density is due to the large parameters chosen for the in-block and between-block probabilities.
We see the degree distribution for each dataset in Figure 13. A strict power-law degree distribution would look like a decreasing straight line in the log-log scale, though in practice such a strict distribution cannot be expected. Thus we can categorize the datasets in the following way: Cora, Pubmed, Airport, and HRG all have a distribution resembling a power-law. For the SBM the distribution clearly does not resemble power-law. For Disease, Road, and LFR, this is not immediately clear due to the scale. However, the same

Dataset	Number of nodes	Density (scale 1e-4)
Cora	2708	14.3
Pubmed	19717	2.3
Airport	3188	36.7
Disease	2665	7.5
Road	5160	5.3
HRG	1000	72.9
LFR	1000	349.8
SBM	1040	2432.2

Table 2: Network size and density for each dataset.



(a) Cora (3.90) (b) Pubmed (4.47) (c) Airport (11.69)

(d) Disease (2.00) (e) Road (2.72) (f) HRG (7.29)

(g) LFR (34.95) (h) SBM (126.12)

Figure 13: Degree distribution for each dataset in the form of scatter plot with log-log scale. The horizontal axis indicates the degree, the vertical axis indicates the number of nodes in that degree. The number in parentheses after each dataset name indicates the average degree in the graph. All plots have the same boundaries of the horizontal axis but the boundaries of the vertical axis vary.

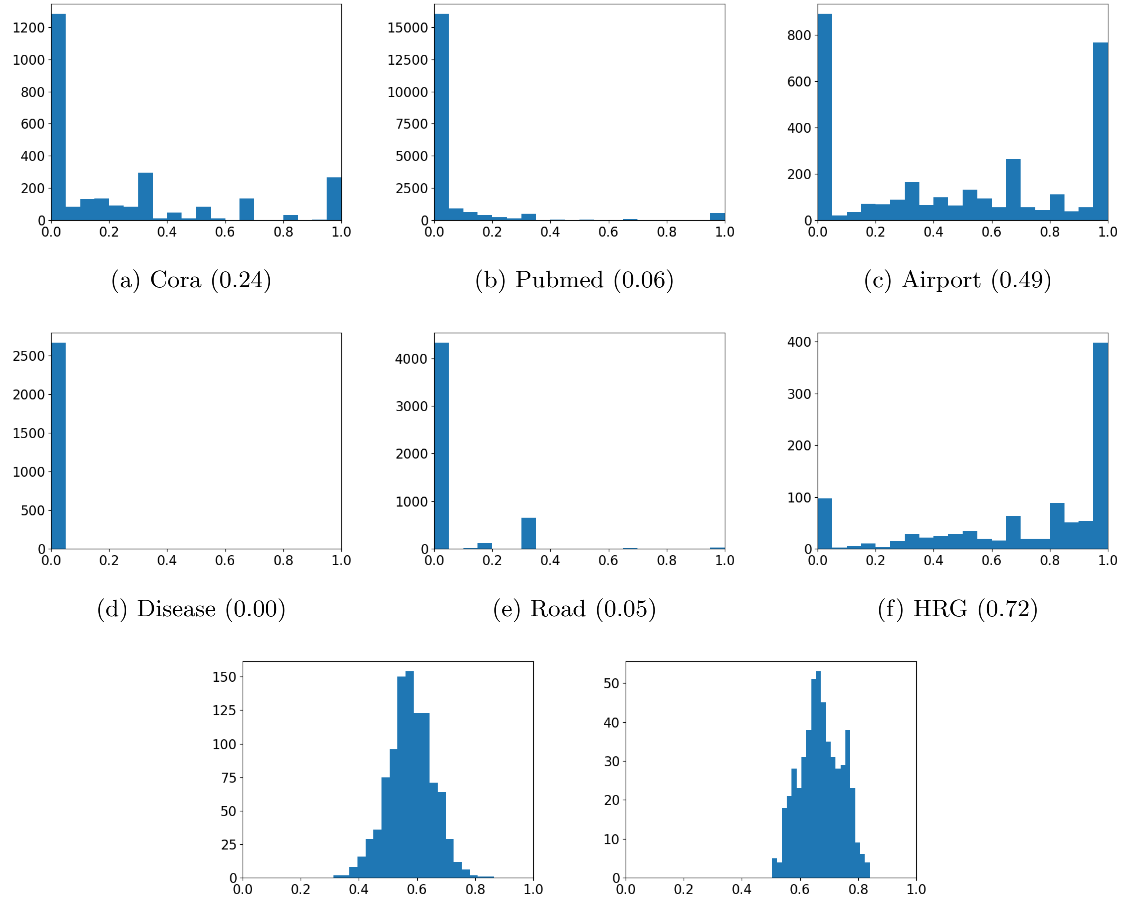
distributions plotted in linear scale (not pictured) lack the right-skewed shape, thus we conclude that they are not power-law distributed.

The average degree is the lowest for Disease and Road and highest for LFR and SBM, which is to be expected given the nature of the respective datasets. For LFR and SBM this is again due to the parameters we have chosen. The average degrees of the other datasets are comparable.

As far as clustering goes (Figure 14), we can divide the datasets in several groups: (mostly) low clustering (Cora, Pubmed, Disease) both high and low clustering (Airport); (mostly) high clustering (HRG), and medium clustering (LFR, SBM). The average clustering of each dataset matches this description as well, even though naturally there are some variations (for example Cora (0.24) and Pubmed (0.06)).

None of the degree-clustering plots (Figure 15) indicate a strict inverse-proportional relationship between degree and clustering. However, some of the datasets (Cora, Pubmed, Airport, HRG) clearly show that high clustering occurs only on low-degree nodes. Stated differently: high degree implies low clustering. We cannot make such observations for Disease (no clustering is present) or Road (the degrees are very low). Similarly, for SBM and LFR we observe no clear relationship between degree and clustering.

The results of the δ-hyperbolicity show a few things. First of all there is some relationship between δ_{worst} and δ_{avg}, in that δ_{avg} appears to be between 5 and 20 times lower than δ_{worst}, or around one order of magnitude. The notable exception here is Cora, where δ_{worst} is more than 40 times higher than δ_{avg}. In



(a) Cora (0.24)

(b) Pubmed (0.06)

(c) Airport (0.49)

(d) Disease (0.00)

(e) Road (0.05)

(f) HRG (0.72)

(g) LFR (0.58)

(h) SBM (0.67)

Figure 14: Clustering coefficient distribution for each dataset in the form of histogram. The horizontal axis indicates the clustering coefficient, the vertical axis indicates the number of nodes in that interval of clustering coefficient. The number in parentheses after each dataset name indicates the average clustering coefficient in the graph. The horizontal axis in each plot is shown from 0 to 1, the vertical axis varies.

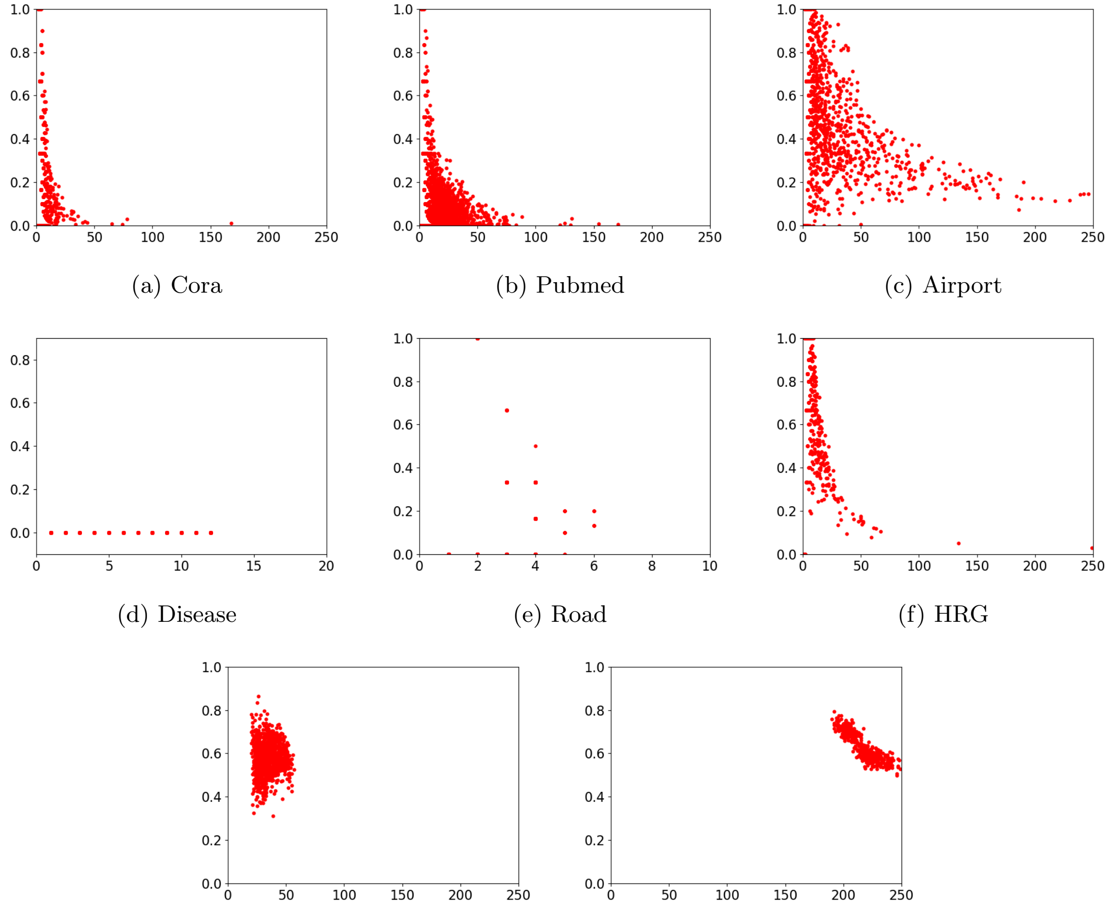
fact, with other datasets the trend is that higher δ_{worst} and higher δ_{avg} go hand in hand, but Cora has lower δ_{avg} than Pubmed but a much higher δ_{worst}. This indicates that the graph of Cora has regions that are not hyperbolic (for example grid-like), but overall it is quite hyperbolic.

As the chosen size and number of blocks of the SBM might affect the calculated δ-hyperbolicity, we calculated it on similar, but larger (up to 20 blocks) SBM graphs. The result is still never higher than 2.0.

A note about the calculations of δ-hyperbolicity: even with 500000 epochs the values reported by Chami et al. could not be reproduced for Cora and Pubmed. According to our calculations both are 2.5-hyperbolic, but this could be because of the smaller number of iterations. Thus it is important to note that the other calculations might not be accurate either. Furthermore, the calculated value for Airport is 1.5, which is higher than the value given by Chami et al.

That being said, we can again categorize the datasets based on this property. SBM, HRG, Airport, and Disease have low hyperbolicity, with Disease being a tree. This is taking into account both the worst-case and average-case δ-hyperbolicity. Cora on the other hand has high δ_{worst} but its δ_{avg} is comparable to the other datasets.

Looking at the Balanced Forman curvature (Figure 16) we again see large similarity between Cora, Pubmed, Airport, and HRG. Each is shaped like a right-skewed bell curve with negative mean, with Pubmed having



(a) Cora

(b) Pubmed

(c) Airport

(d) Disease

(e) Road

(f) HRG

(g) LFR

(h) SBM

Figure 15: Degree-clustering plots for each dataset. The horizontal axis indicates the degree of a node, the vertical axis indicates the clustering coefficient of that node. Both axes are scaled the same for all plots, except for Disease and Road to make the plots more readable.

notable a spike around 0. On the other hand, Disease, LFR, and SBM have very similar distributions with most edges around 0 and some close to -2, with average around 0. Road looks similar to the other bell curve distributions but less smooth and closer to 0. We categorize the datasets' Balanced Forman curvature into negative (Cora, Pubmed, Airport, HRG) and close to 0 (Disease, Road, LFR, SBM).
A summary of these results can be found in Table 5. Here we only look at each property in a broad sense, looking at the trend. For example, we consider a network to have a power-law degree distribution if the log-log degree distribution plot resembles a line. We also do not include less relevant properties (average degree, size, density) in this table but they should still be kept in mind.

4.1.2 Model performance

Next we present the performance of both models on each dataset with the previously found hyperparameters. The results can be seen in Table 4. The results of Cora and Disease presented by Chami et al. [1] differ significantly from ours, with them achieving a much smaller error reduction on Cora and a much larger error reduction on Disease. It is unclear where this disparity comes from but it will be taken into account when analyzing these results.
On Road we see an error reduction of 25%, which is quite low compared to the other real-world datasets. On the synthetic datasets, HRG, LFR, and SBM, we notice a trend: HRG, being purely hyperbolic, has

Dataset	δ_{worst}	δ_{avg}	Ratio
Cora	11.0	0.26	42.3
Pubmed	3.5	0.36	9.7
Airport	1.0	0.17	5.9
Disease	0.0	0.0	-
Road	20.0	2.59	7.7
HRG	1.0	0.06	16.7
LFR	1.5	0.22	6.8
SBM	1.0	0.08	12.5

Table 3: δ-hyperbolicity of datasets. Cora, Pubmed, Airport, Disease are taken from Chami et al. [1], others are calculated with 50000 samples. The ratio shown is $\delta_{worst}/\delta_{avg}$.

Dataset	GCN	HGCN	Error reduction
Cora	87.4 ± 2.5	93.1 ± 0.4	-45.2%
Pubmed	91.1 ± 0.5	96.3 ± 0.0	-58.4%
Airport	91.5 ± 0.2	94.8 ± 0.4	-38.8%
Disease	56.9 ± 2.2	63.9 ± 0.1	-16.2%
Road	96.4 ± 0.1	97.3 ± 0.1	-25.0%
HRG	86.8 ± 1.9	97.1 ± 0.2	-78.0%
LFR	90.1 ± 0.2	91.2 ± 0.4	-11.1%
SBM	92.4 ± 0.2	92.6 ± 0.3	-2.6%

Table 4: Model performance of GCN and HGCN on each dataset. For Pubmed the result is taken from Chami et al. [1].

the largest error reduction observed so far, whereas SBM, which we know has no underlying hyperbolicity, shows almost no difference in model performance. On the LFR the difference is still small, but larger than that of the SBM.

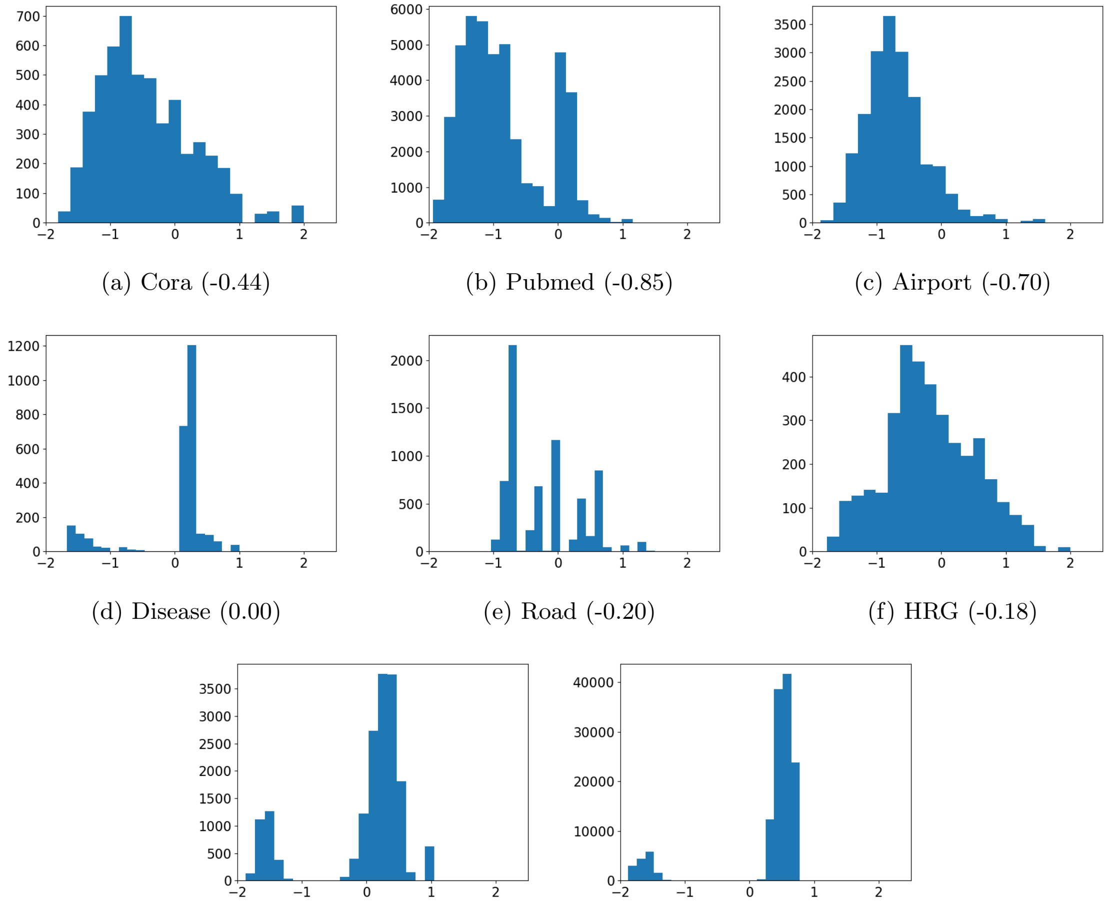
In Table 5 the dataset properties are summarized together with the difference in model performance. To make things simpler, we categorize error reduction of 30% percent or more as high, lower than 30% as low, and lower than 5% as very low.

Dataset	Degree distr.	Clustering	Degree-clust.	δ-hyperbolicity	BF curv.	Error red.
Cora	Power-law	Mostly 0	Hd - Lc	High worst, low avg.	Negative	High*
Pubmed	Power-law	Mostly 0	Hd - Lc	Medium	Negative	High
Airport	Power-law	Mostly 0 and 1	Hd - Lc	Low	Negative	High
Disease	-	Only 0	-	Low (tree)	0	Low*
Road	-	Mostly 0	-	High	0, varying	Low
HRG	Power-law	Mostly 1	Hd-Lc	Low	Negative	High
LFR	-	Medium	-	Low	0	Low
SBM	-	Medium	-	Low	0	Very low

Table 5: Summary of dataset properties. “Hd-Lc” stands for high degree - low clustering and indicates a present relation between the two. The last column shows the error reduction of the HGCN as opposed to the GCN, where * in the last column indicates that this result does not match that of Chami et al. [1]. Size, density, average degree, and average density are omitted.

4.1.3 Robustness to noise

To analyze both models' robustness to noise, we run experiments on the HRG. We vary the temperature (noise on edges) and the noise on the features. We report the ROC AUC on the test set over 5 runs with



(a) Cora (-0.44)

(b) Pubmed (-0.85)

(c) Airport (-0.70)

(d) Disease (0.00)

(e) Road (-0.20)

(f) HRG (-0.18)

(g) LFR (-0.02)

(h) SBM (0.28)

Figure 16: Balanced Forman curvature distribution for each dataset in the form of histograms with 20 bins. The horizontal axis represents the curvature, the vertical axis is the number of edges in that interval. The number in parentheses after each dataset name indicates the average value over all edges. The horizontal axis is scaled from -2 to 2.5 for all plots (no larger value is observed), the vertical axis varies.

different random seeds. The results are summarized in Table 6. In general, results show that the higher noise, the worse the performance, for both models and both types on noise. However there are a few noteworthy exceptions.

Low feature noise ($\sigma = 0.2$) significantly improves the GCN performance, which is not the case for the HGCN. This is true for temperature up to 0.6. On the other hand, low temperature ($T = 0.2$) increases performance of HGCN slightly, which is not the case for the GCN. This trend is present for low (0.0 and 0.2) and very high (0.8 and random) feature noise.

The last column results in around 0.5 ROC AUC for both models, which indicates that the guesses are random in this case. In general, we observe that in all cases, except random edges, the HGCN outperforms the GCN, though this gap decreases significantly when both types of noise are strongly present.

4.2 Learnable curvature

Next we investigate the learnable curvature of the HGCN. We start by looking at the training of the learnable curvature parameter of each layer of the neural network, in Figure 17, Figure 18, and Figure 19. Only a selection of the plots are shown here, the rest can be found in Appendix B. The plots are achieved by

		Temperature					
	HGCN	0.0	0.2	0.4	0.6	0.8	Random
Feature noise	0.0	97.1	$\underline{97.5}$	95.7	89.4	84.3	49.9
	0.2	95.6	$\underline{95.8}$	93.3	86.8	82.6	50.1
	0.4	92.5	92.2	88.8	85.0	78.4	49.9
	0.6	92.5	90.9	84.2	79.4	73.0	49.9
	0.8	86.9	$\underline{87.8}$	82.1	74.6	69.3	50.1
	Random	80.0	$\underline{80.5}$	73.0	68.9	64.0	49.9

		Temperature					
	GCN	0.0	0.2	0.4	0.6	0.8	Random
Feature noise	0.0	87.0	81.1	81.1	75.6	73.1	50.3
	0.2	$\underline{91.2}$	$\underline{89.9}$	$\underline{86.1}$	$\underline{81.0}$	68.6	49.9
	0.4	83.0	$\underline{83.2}$	77.6	68.2	66.7	50.2
	0.6	80.3	78.8	75.6	$\underline{69.3}$	66.2	50.3
	0.8	79.3	76.0	68.8	65.5	64.2	49.8
	Random	74.9	$\underline{76.5}$	$\underline{69.8}$	61.4	62.3	49.6

Table 6: Experiments with noise of HGCN and GCN on the HRG. Underlined values indicate that the respective accuracy is higher than a value up or to the right of in the table.

recording the curvature in each layer at each training epoch. This is done for 3 different random seeds and the two runs that exhibit most differences are collected (meaning differences with regard to shape of the curves, order of the layers, etc.). The goal is to capture variability in the training of the same dataset. With the "order" of the layers what is meant is whether the curvature increases with each following layer, or decreases, or some other pattern.

First, as a general observation we notice that the learnable curvature rarely converges during training. For example, in the case of Cora (Figure 17), the values appear to increase almost constantly. Further experiments (not pictured) show that even when training for up to 5000 epochs, the values rarely converge. In fact, in some cases the trend changes after a long number of epochs (e.g. one of the values increases for 3000 epochs and then starts decreasing). The same can be said about the average of the curvatures. Futhermore, this happens for different types of datasets.

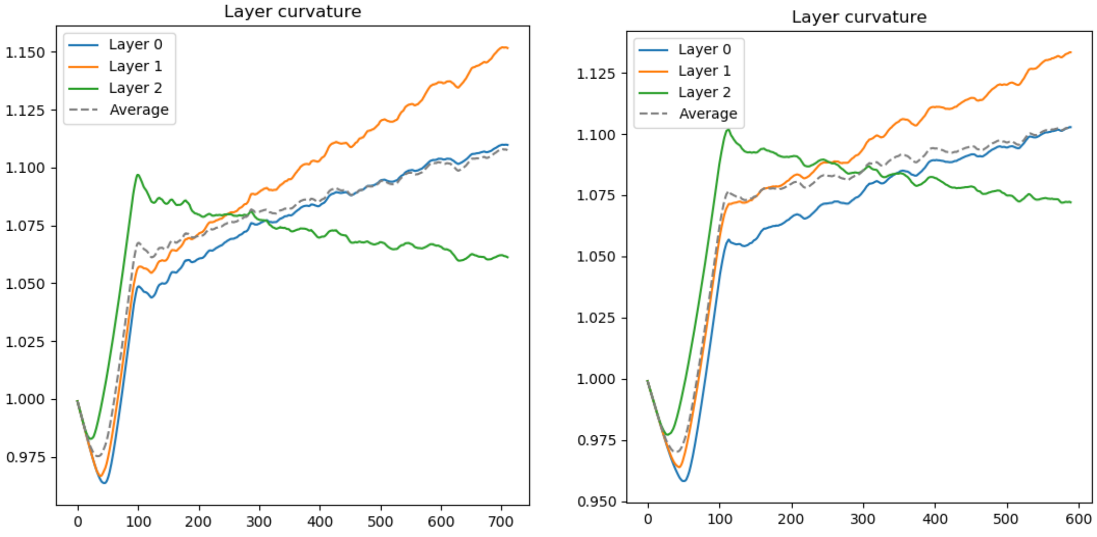
More specifically, we observe that in some cases such as Cora (Figure 17) the training looks almost the same for different runs, with only some small differences in values. In other cases, such as Disease (Figure 18), the process looks similar, but in the end the order of the curvatures is different. Lastly, in SBM (Figure 19) we see that the progression looks completely different between runs. In general, there does not seem to be consistency between different runs, at least not for all datasets.

In Table 7 we see the final learned values for each layer (corresponding to run 1 of each of the plots). Here we further observe that the average final curvature is around 1 for all datasets except HRG. In fact, HRG exhibits the largest values (in layers 0, 2, and the average). There appears to be no trend for progression of curvature over the layers. For example, in Airport we see that layer 2 has the lowest and layer 1 the highest final value, but in the case of HRG this is the opposite.

Next we perform the same experiments only on the HRG, with different values of α. The parameter α controls the power of the power-law distribution of the degrees. We take 3 values for α: 0.6 (low), 0.75 (medium), 0.9 (high). The results are summarized in Table 8. We notice that for layers 0 and 2, as well as the average, increasing α decreases the learned value. For layer 1 the opposite is true, the value increases as α increases.

4.3 Embedding dimension

In order to find out if the performance of the GCN can match that of the HGCN, we take two datasets: HRG and SBM. This is because these are very different: HRG is hyperbolic and the HGCN has been shown to perform significantly better, whereas the SBM is not hyperbolic and both models have similar performance



(a) Run 1

(b) Run 2

Figure 17: Learnable curvature optimization on Cora. The horizontal axis shows the epoch and the vertical axis shows the value of the curvature at each layer at that epoch.

Dataset	Layer 0	Layer 1	Layer 2	Average
Cora	1.11	1.15	1.06	1.11
Airport	1.00	1.40	0.78	1.06
Disease	1.52	1.37	0.13	1.01
Road	0.99	1.07	1.11	1.06
HRG	2.22	0.8	4.33	2.45
LFR	1.26	1.1	1.11	1.00
SBM	1.85	0.7	0.35	0.97

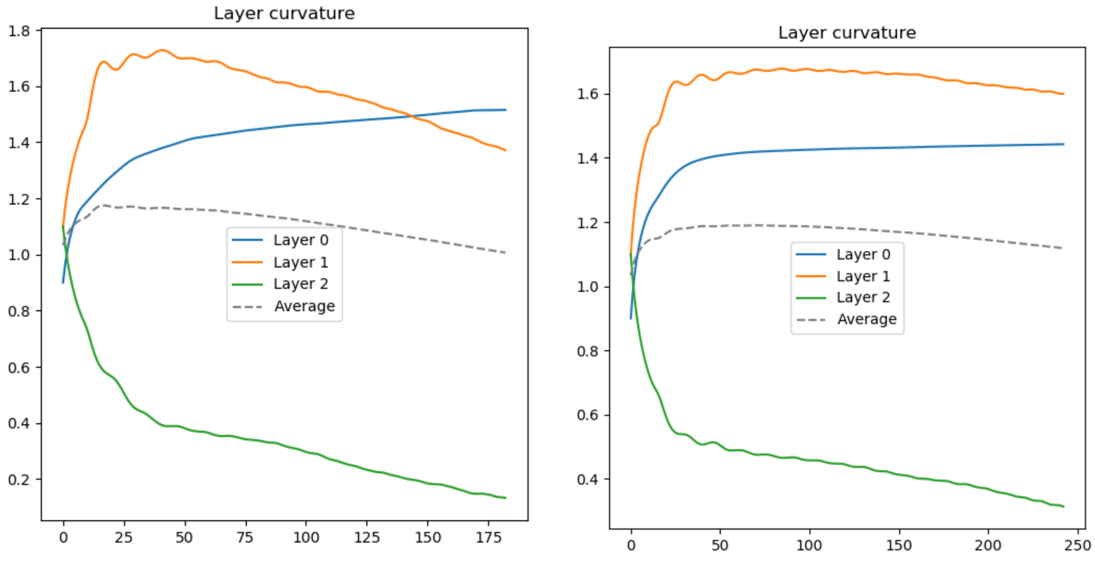
Table 7: Final learned curvature in each layer for each dataset.

on it. Furthermore, we vary the embedding dimension, while keeping the number and sizes of hidden layers the same (2 hidden layers of dimension 16). As embedding dimensions we take 2 (low), 16 (medium), 128 (high), 1024 (very high).

In the case of HRG (Table 9) we see that the relative performance gap (taking into account error reduction) stays consistent with the dimension. It is still noteworthy that the accuracy of the HGCN increases only from 93.2 to 97.4, whereas the GCN goes from 74.1 to 91.3, so the absolute difference is much higher in the GCN's case. Furthermore, we see that while the GCN's performance steadily increases with the dimension, for the HGCN there is barely an increase beyond dimension 16.

Nonetheless, even with embedding dimension 1024, the HGCN still significantly outperforms the GCN. In fact, the GCN at 1024 dimensions still gives a worse result than the HGCN at 2 dimensions.

The results for the SBM are quite different, as shown in Table 10. The HGCN appears to achieve the same results regardless of the embedding dimension. The GCN on the other hand shows some increase in performance when the embedding dimension becomes higher. In any case the results are very close and the variance is significant, but it looks like for a higher dimension (128 or 1024), both models perform about the same. At dimension 2 the HGCN shows a better result by a small margin.



(a) Run 1

(b) Run 2

Figure 18: Learnable curvature optimization on Disease. The horizontal axis shows the epoch and the vertical axis shows the value of the curvature at each layer at that epoch.

α value	Layer 0	Layer 1	Layer 2	Average	Accuracy
0.6	2.78	0.16	6.22	3.05	94.8
0.75	2.22	0.8	4.33	2.45	97.0
0.9	1.8	1.38	2.75	1.98	98.1

Table 8: Final learned curvature in each layer for HRG with changing α values.

5 Discussion

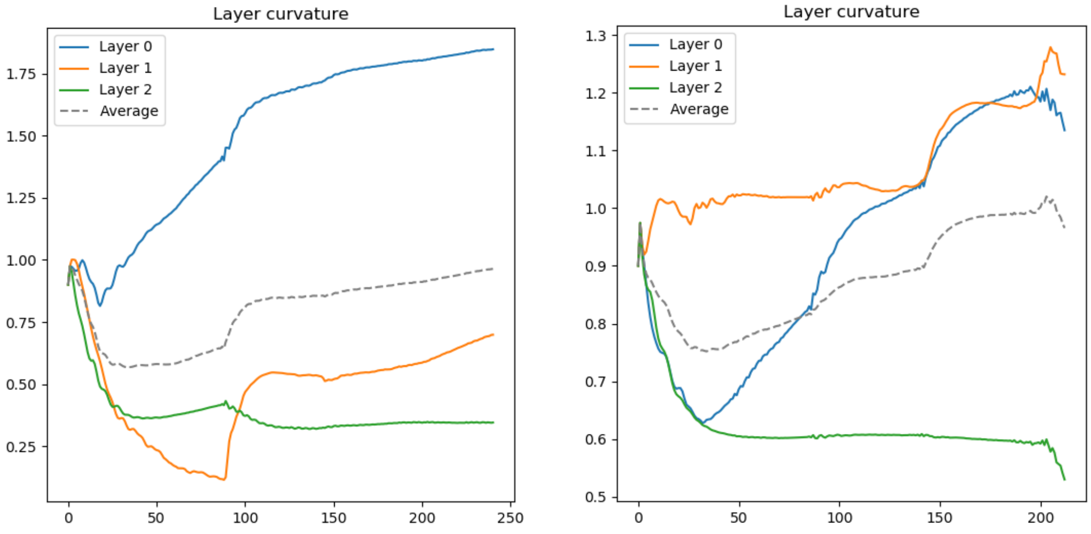
In this section we interpret the results of the previous experiments and use them to answer the research questions. We also explain their limitations and link to existing research in the field, as well as giving a general discussion with possible directions for further research.

5.1 Research question 1: What properties of the dataset guarantee that HGNNs will perform better than (Euclidian) GNNs?

To answer the first research question we started by looking at what dataset properties correlate with differences in model performance of the HGCN and GCN. We do not specifically consider size and density as predictors of performance, as they are not theoretically linked with hyperbolicity. But they will be taken into account when explaining differences in resutlts.

The traditional graph features related to hyperbolicity give some indication of model performance. The power-law degree distribution and high-degree-low-clustering relationship appear together every time, and they are a good indicator that the HGCN will perform better than the GCN. Thus one of them can be omitted in future studies on such properties, for example only looking at degree distribution since it is more directly related to hyperbolicity. It also has to be noted that both of these relationships can only exist when high degrees are present, which is not the case in Disease for example. It is also clear, however, that degree distribution is not necessary for hyperbolicity, as Disease does not have any distinct distribution due to having low degrees in general. Despite this, Disease does have underlying hyperbolic geometry, which is reflected in the performance of both models (taking into account the results of Chami et al. [1]).

Another direction that could be taken with degree distribution is determining the power of the power-law



(a) Run 1

(b) Run 2

Figure 19: Learnable curvature optimization on SBM. The horizontal axis shows the epoch and the vertical axis shows the value of the curvature at each layer at that epoch.

Dim.	GCN	HGCN	Error reduction
2	74.1 ± 3.4	93.2 ± 2.1	-73.7%
16	86.8 ± 1.9	97.1 ± 0.2	-78.0%
128	89.0 ± 1.3	97.3 ± 0.3	-75.5%
1024	91.3 ± 1.0	97.4 ± 0.4	-70.1%

Table 9: Performance on HRG of each model for a given embedding dimension

distribution. This power is related to the curvature of the graph, at least in HRG [62]. We shortly investigated this by altering the α parameter of the HRG generation, showing that higher α results in better performance. This shows that there could well be a relation between this power (or how skewed the degree distribution is) and the preference for embedding in hyperbolic space. This approach does, however, raise new difficulties, like what is the best way to fit a power-law on the distribution of a real-world dataset? Furthermore our short experiment does not show the performance gap of the HGCN and GCN, which would be more useful.

It is interesting to note that the LFR dataset we created does not exhibit power-law degree distribution, despite being generated by a model that should include it. This is likely due to the chosen parameters, as well as the relatively small size of the graph. Despite the lack of power-law degree distribution, the performance gap between GCN and HGCN is notably larger on the LFR than the SBM. This indicates that the small amount heterogeneity of the degrees in the LFR still contributes to some underlying geometry which can be captured by the HGCN. It still needs to be kept in mind that such small differences could be the consequence of the variability of the results, or even other differences in the graphs themselves, like the number of blocks. Clustering varies quite a bit in the datasets where a heterogeneous degree distribution is observed. Cora and Pubmed exhibit 0 clustering for the majority of nodes, HRG has the opposite, and Airport is a mix of both. However this does not seem to relate directly to performance, so we can attribute this to local differences in the networks in the smaller-degree nodes. HRG has a tendency for clustering, regardless of temperature. It may very well be that airport networks as a whole have some tendency to cluster locally, whereas this does not happen in citation networks. However this difference in local clustering should not affect the global hyperbolic structure. Furthermore, while clustering has been observed in many scale-free and hierarchical

Dim.	GCN	HGCN	Error reduction
2	92.1 ± 0.2	92.7 ± 0.4	-7.6%
16	92.4 ± 0.2	92.6 ± 0.3	-2.6%
128	92.8 ± 0.1	92.6 ± 0.2	+2.8%
1024	92.5 ± 0.3	92.8 ± 0.0	-4.0%

Table 10: Performance on SBM of each model for a given embedding dimension.

graphs, it is not necessary for their existence. Therefore, it seems that clustering does not have utility for predicting hyperbolicity and thus model performance.

Regarding δ-hyperbolicity, Chami et al. [1] conclude that the performance of the HGCN, and more specifically its advantage over Euclidian models, is directly tied to low δ-hyperbolicity. However, our results indicate that this may not always be true. The stochastic block model (SBM), for which we know there is no underlying hyperbolic geometry (as opposed to for example HRG), has low δ-hyperbolicity even when increasing the graph size and the number of blocks. This is confirmed by the traditional graph features that we investigated: no power-law degree distribution, medium clustering, no clear relationship between degree and clustering. Furthermore, the performance of both models on the SBM dataset is almost the same.

This outcome likely relates to the fact that the δ-hyperbolicity of a clique is 0, the same as a tree. Real-world datasets are often not very dense, so this is not a problem, but the SBM shows that cases may occur where the δ-hyperbolicity is low despite no underlying hyperbolic geometry. It should be noted that the results for δ-hyperbolicity may not be accurate as they are calculated by sampling, but this is another inherent limitation of this metric.

To deal with this inconsistency, we propose the following procedure: when the density of the graph is low, simply check the δ-hyperbolicity to decide what model to use. If the graph is very dense, instead consult the degree distribution to check for heterogeneity/power-law.

We further added δ_{avg}, the average-case δ over the taken samples, as a dataset property. Overall it appears that δ_{avg} has a close relationship with δ_{worst} (which is the classical definition of δ-hyperbolicity), with around one order of magnitude difference. The notable exception here is Cora, where δ_{avg} is quite low despite δ_{worst} being quite high. Considering we expect Cora to have a similar hyperbolicity as Pubmed, as they are both citation networks, it would make more sense to look at the average hyperbolicity. This would deal with the inherent problem of δ-hyperbolicity, specifically δ_{worst}, which is that is can be determined entirely by a small part of the graph which is very non-hyperbolic (for example ring or grid). It could be that in Cora there is such a region, which is affecting δ_{worst} despite having a small effect on the overall hyperbolicity. Thus using δ_{avg}, or at least taking it into account, makes more sense. Furthermore, one could instead look at the distribution of the found δ, getting an even clearer picture.

The Balanced Forman curvature is closely related to degree distributions on the datasets we used. This means that it is in general a useful indicator but still does not capture the hyperbolicity of Disease, being a tree. This is likely because Disease contains many edges connected to leaf nodes, which can be shown to have positive curvature. Such leaf edges would not necessarily affect the geometry of the graph, but there could be a large number of them. So much so that the distribution of edge curvature does not anymore give a clear picture of the underlying curvature of the graph. A way to get past this in general would be to distinguish between different types of edges, but it is not obvious how this could be done for an arbitrary graph. Thus we conclude that Balanced Forman curvature does not have utility as a global measure for predicting the performance of HGNNs, and is better left as a local measure of curvature for other tasks. It does appear to capture scale-free graphs to some extent - Cora, Pubmed, Airport, and HRG all have similar degree distributions and similar Balanced Forman curvature distributions. Nonetheless, in such cases it is still more useful to look at degree distribution, as it is easier to compute.

To summarize, our results indicate that when given a dataset and considering whether to use a HGNN or an Euclidian GNN, the best thing would be to take into account multiple properties. If the dataset has low density, then δ-hyperbolicity, specifically δ_{avg} indicates that an HGNN would be a better choice. If the density is high, one should look at the degree distribution to determine hyperbolicity.

More experiments are necessary to make these findings more concrete and to specify exact values of the metrics we discussed. Furthermore, it would be interesting to investigate whether other values used in literature (such as sectional curvature) also have utility for this task.

To further investigate the first research question, we observed how the performance of both models is affected by noise, both in the features and in the graph structure. In general, we observe that adding any type of noise, regardless of the model, makes the result worse. This was to be expected but there are some exceptions to this trend that should be discussed. The GCN improves with some noise in the features present. This likely relates to the fact that the Euclidian embedding space is not well suited for the original data, and adding noise makes it more "flat", at least with regards to the features. This trend also appears even when some temperature is present, implying even further that the hyperbolic features are what the GCN has a problem capturing.
It is also noteworthy that the same trend is not present in the HGCN, where the quality of the predictions strictly decreases when adding noise to the features. The HGCN on the other hand slightly improves when noise in the edges, or temperature, is present. One reason for this could be that the message passing of the GNN architecture is more effective when there are more connections in the graph. However, this would also apply to the GCN. The more likely explanation is that the HGCN is able to make good use of the node features and thus infer where the clustering caused by temperature will appear.
It still needs to be kept in mind that despite these reductions, the HGCN performs better in every single case. Thus, even though the GCN reacts better to noise in the features, the HGCN is a better choice regardless, if the goal is to have the highest accuracy possible. It would also be interesting to see whether the same trend is present when the data is not hyperbolic, for example on Road, or on other datasets in general.
Finally, both models achieve an accuracy of around 0.5 when the edges are completely random. This is reasonable, as the task is link prediction, which cannot be done in a meaningful way if the edges are random.

5.2 Research question 2: What role does the curvature play as a learnable parameter in the HGCN?

Next we observed the curvature of each layer as a learnable parameter. The lack of convergence in most cases and the lack of structure in the learned parameters indicate that the HGCN does not strictly relate the curvature of the data to these parameters. If the opposite were the case, we would expect that the learned values would always converge in the same way, at least when there is hyperbolic geometry present in the data.
This theory is supported by the fact that comparing the learned curvature between datasets does not seem to have a strong correlation to their respective properties and hyperbolicity. Most of the values are around 1, even for datasets where it is known that the underlying geometry is very different (for example Disease and SBM). We do notice that for the HRG the values, including the average, are the highest of all datasets. This is consistent with the fact that HRG is very hyperbolic, so the learned curvature might still be related to curvature in the data. However, even if this is the case, this correlation does not appear to be useful for predicting the learned values from the data.
The experiments with changing the α value of the underlying curvature, which directly relates to the amount of curvature present in the dataset [25], show the same thing. The curvature of layers 0 and 2, as well as the average, decreases as α increases. Only for layer 1 the trend makes sense, as higher α relates to higher learned curvature.
Despite some observations, we can reasonably conclude that the learned curvature does consistently correlate to the dataset used and its properties. Just as Chami et al. show, the curvature of each layer serves to numerically stabilize the model. As such, the best strategy is to find what value of curvature leads to the best results, by setting it as a learnable parameter like Chami et al., or by other methods like reinforcement learning [46]. It could be interesting to experiment with setting the curvature to different pre-decided values, possibly even a different value in each layer. However, based on our findings, this would not be a very useful new direction for research.

5.3 Research question 3: Can the performance gap between HGNN and GNN be compensated for by increasing the embedding dimension of the GNN?

Experimenting with changing the number of embedding dimension of both models provides more insight into their usefulness. When the data is hyperbolic, the HGCN benefits from a low embedding dimension. This is shown in our experiments as well as in existing literature [2]. Despite the relative error reduction between the HGCN and GCN being very close for any embedding dimension, the absolute difference should also be considered. At very low embedding dimension (specifically 2) the HGCN still provides a high quality prediction, whereas that of the GCN is quite poor.

Both the HGCN and GCN's performances are significantly improved when increasing the dimension from 2 to 16, but beyond 16 dimensions the HGCN barely improves, whereas the GCN increases steadily. These observations tell us that the GCN really needs a higher embedding dimension to make a good prediction. On the other hand, the HGCN does quite well even with a very low embedding dimension, and it does not benefit much from having a higher one. This is consistent with the theory that hyperbolic/scale-free data can be embedded in a lower dimension by an HGNN.

We also notice that even when the GCN can make use of 1024 dimensions, and the HGCN only 2, the HGCN is still better by a small margin. This further solidifies the benefits of HGCN on hyperbolic data, regardless of embedding dimension.

When it comes to non-hyperbolic data like the SBM graph, the performance of both models is much more comparable. However, at low embedding dimension the HGCN still appears to have a slight edge. This could be due to the randomness of the experiments, but more likely the HGCN is simply more flexible due to the learnable curvature parameters, which make a difference at low dimensions.

Just as in other experiments, regardless of the setup, the HGCN is never worse than the GCN. In this case there are some experiments (SBM with 128 dimensions) where the GCN might be slightly better, but this is still negligible and could be due to the inherent randomness.

5.4 General remarks

From all the performed experiments we can make a general observation that the HGCN performs better than the GCN in every single case, regardless of dataset, noise, and embedding dimension. Of course, this difference is much smaller when there is no underlying hyperbolic geometry for the HGCN to capture. This does not mean that any HGNN is better than any Euclidian GNN, as Chami et al. have shown that this is not the case. So this difference in performance gap might become more relevant if different Euclidian or hyperbolic models are chosen.

In any case, based on GCN and HGCN specifically, it seems that simply introducing hyperbolic space to any Euclidian GNN, when possible, will not make the performance worse. In fact, it can be said that utilizing an HGNN over an Euclidian GNN will not significantly worsen performance, whereas it may significantly improve it. As a whole, this indicates that the HGCN not only captures underlying hyperbolic geometry if it is present, but also serves as extra flexibility for the model, in case the data is not hyperbolic.

Knowing this, we also have to take into account the fact that the computations of HGNNs, especially training, take much longer. A concrete study on these times would be a fitting follow-up to this thesis, so that a more informed overview can be made. But in general we can say that using an HGNN on a non-hyperbolic data might not be worthwhile because of higher computation times. On hyperbolic datasets using a HGNN is more reasonable, as the high computation times are compensated by significantly better performance. But in any case, if a high prediction accuracy is not the goal and the priority is low training and computation times, it is advised to simply use a Euclidian GNN. Furthermore, depending on the specific hyperbolic and Euclidian models chosen, this gap can disappear, in which case computation times become the important factor.

Another direction that could have been taken to answer our research questions is investigate hyperbolicity of features: This thesis focused purely on hyperbolicity of the graph structure, not taking into account node features. It is clear that node features also play a role, as hyperbolic neural networks exist [9] that do not even work with graphs. It would be useful to take a look, similarly to this work, into what kind of attributes make node features hyperbolic, if there even are such attributes. We have already made steps towards this by adding noise to the features of the HRG but this just gives a basic idea.

This research focused on a particular variation of a particular HGNN architecture, as well as on a particular task. This was necessary in order to be able to be able to give depth to the research, but it also leaves a lot of possibilities for future work. For example, our experiments can be repeated with added attention, which would mean working HGCN with attention and GAT [19]. The attention mechanism can better capture hierarchies, shown by Chami et al., which could lead to different results. Furthermore, the link prediction task that was the focus of our experiments is only one possibility for a graph learning task. Node classification, as done by Chami et al. [1], and graph classification, as done by Liu et al. [10], are two other tasks with a wide variety in application. For a more complete picture these should also be included.
Using the HGCN might also not be the best option when it comes to fully utilizing hyperbolic geometry. It is one of the more established papers in the field, but has been shown to have many limitations, for example the tangent space operations, or the learnable curvature. More state-of-the-art models, like the ACE-HGNN [46] or the H2H-GCN [12] would be better suited for this. Comparing these performance of these models would also be useful.

6 Conclusion

In this thesis we investigated the performance of HGNNs, specifically the HGCN [1]. The first research question was what properties of a dataset can predict a good performance of an HGNN, compared to that of a Euclidian GNN. We conclude that δ-hyperbolicity is a good predictor but the average-case δ-hyperbolicity may be a better option. Furthermore, δ-hyperbolicity may not be reliable when the dataset is very dense, and in such cases traditional graph properties should be considered, specifically the degree distribution. If the degree distribution is very heterogeneous, i.e. resembling a power-law, the graph can be considered hyperbolic and thus an HGNN should have an advantage. Furthermore, our experiments indicate that Balanced Forman curvature, a measure of Ricci curvature for graph edges, does not appear to have utility for this problem.
We have also shown that, when the given dataset is highly hyperbolic, HGNNs are better equipped to deal with noise in the graph structure, whereas Euclidian GNNs handle noise in the features better. Despite this, the HGCN is a better tool than the GCN when it comes to accuracy, regardless of noise.
The experiments in this thesis also indicate that the learnable curvature is simply an extra parameter in the GNN that serves to improve the expressiveness of the model. Attempts to set it to a given value, based on dataset properties for example, will likely not lead to optimal results. The recommended way to deal with this is to learn the curvature, with one of the existing methods or possibly a new one. It also appears that the inclusion of curvature as a learnable model parameter gives the model extra expressiveness, regardless of geometry of the data.
The last part of this work confirmed the known fact that HGNNs benefit from low embedding dimension when the dataset used is hyperbolic, but we have also shown that even with non-hyperbolic data there is still some benefit to using HGNNs when the dimension is low, further confirming the previous point about the learnable curvature as increasing expressiveness regardless of curvature.
In conclusion, this thesis has made progress into developing a better understanding of HGNNs and when they are advantageous. Our research has identified some dataset properties are good predictors of performance, confirmed the role of curvature values in training hyperbolic models, and related model performance to embedding dimension.

References

[1] I. Chami, R. Ying, C. Re, and J. Leskovec, “Hyperbolic graph convolutional neural networks,” in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, no. 438, pp. 4868–4879, Red Hook, NY, USA: Curran Associates Inc., Dec. 2019.

[2] M. Yang, M. Zhou, Z. Li, J. Liu, L. Pan, H. Xiong, and I. King, “Hyperbolic Graph Neural Networks: A Review of Methods and Applications,” Feb. 2022. arXiv:2202.13852 [cs].

[3] T. N. Kipf and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks,” Feb. 2017. arXiv:1609.02907 [cs, stat].

[4] P. Veličković, "Everything is connected: Graph neural networks," *Current Opinion in Structural Biology*, vol. 79, p. 102538, Apr. 2023.

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 4–24, Jan. 2021. arXiv:1901.00596 [cs, stat].

[6] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges," May 2021. arXiv:2104.13478 [cs, stat].

[7] T. S. Cohen, M. Geiger, J. Koehler, and M. Welling, "Spherical CNNs," Feb. 2018. arXiv:1801.10130 [cs, stat].

[8] W. Peng, T. Varanka, A. Mostafa, H. Shi, and G. Zhao, "Hyperbolic Deep Neural Networks: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 10023–10044, Dec. 2022. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[9] O. Ganea, G. Becigneul, and T. Hofmann, "Hyperbolic Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018.

[10] Q. Liu, M. Nickel, and D. Kiela, "Hyperbolic graph neural networks," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, no. 739, pp. 8230–8241, Red Hook, NY, USA: Curran Associates Inc., Dec. 2019.

[11] W. Chen, X. Han, Y. Lin, H. Zhao, Z. Liu, P. Li, M. Sun, and J. Zhou, "Fully Hyperbolic Neural Networks," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Dublin, Ireland), pp. 5672–5686, Association for Computational Linguistics, May 2022.

[12] J. Dai, Y. Wu, Z. Gao, and Y. Jia, "A Hyperbolic-to-Hyperbolic Graph Convolutional Network," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 154–163, June 2021. ISSN: 2575-7075.

[13] J. Sun, Z. Cheng, S. Zuberi, F. Perez, and M. Volkovs, "HGCF: Hyperbolic Graph Convolution Networks for Collaborative Filtering," in *Proceedings of the Web Conference 2021*, WWW '21, (New York, NY, USA), pp. 593–601, Association for Computing Machinery, June 2021.

[14] R. Sawhney, S. Agarwal, A. Wadhwa, and R. Shah, "Exploring the Scale-Free Nature of Stock Markets: Hyperbolic Graph Learning for Algorithmic Trading," in *Proceedings of the Web Conference 2021*, WWW '21, (New York, NY, USA), pp. 11–22, Association for Computing Machinery, June 2021.

[15] J. Leskovec, "Machine Learning with Graphs, Stanford lecture series," 2021.

[16] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online Learning of Social Representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, Aug. 2014. arXiv:1403.6652 [cs].

[17] A. Grover and J. Leskovec, "node2vec: Scalable Feature Learning for Networks," July 2016. arXiv:1607.00653 [cs, stat].

[18] M. Bronstein, "Beyond Message Passing: a Physics-Inspired Paradigm for Graph Neural Networks," May 2022. testing.

[19] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," Feb. 2018. arXiv:1710.10903 [cs, stat].

[20] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), pp. 1025–1035, Curran Associates Inc., Dec. 2017.

[21] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, “Simplifying Graph Convolutional Networks,” in *Proceedings of the 36th International Conference on Machine Learning*, pp. 6861–6871, PMLR, May 2019. ISSN: 2640-3498.

[22] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How Powerful are Graph Neural Networks?,” Feb. 2019. arXiv:1810.00826 [cs, stat].

[23] F. Schuller, “Lectures on the Geometric Anatomy of Theoretical Physics,” 2015.

[24] J. M. Lee, *Riemannian Manifolds*, vol. 176 of *Graduate Texts in Mathematics.* New York, NY: Springer, 1997.

[25] D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguñá, “Hyperbolic geometry of complex networks,” *Physical Review E*, vol. 82, p. 036106, Sept. 2010. Publisher: American Physical Society.

[26] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, “Understanding oversquashing and bottlenecks on graphs via curvature,” Nov. 2022. arXiv:2111.14522 [cs, stat].

[27] G. Bachmann, G. Becigneul, and O. Ganea, “Constant Curvature Graph Convolutional Networks,” in *Proceedings of the 37th International Conference on Machine Learning*, pp. 486–496, PMLR, Nov. 2020. ISSN: 2640-3498.

[28] R. Sarkar, “Low Distortion Delaunay Embedding of Trees in Hyperbolic Plane,” in *Graph Drawing* (M. van Kreveld and B. Speckmann, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 355–366, Springer, 2012.

[29] A. B. Adcock, B. D. Sullivan, and M. W. Mahoney, “Tree-Like Structure in Large Social and Information Networks,” in *2013 IEEE 13th International Conference on Data Mining*, pp. 1–10, Dec. 2013. ISSN: 2374-8486.

[30] W. S. Kennedy, O. Narayan, and I. Saniee, “On the Hyperbolicity of Large-Scale Networks,” June 2013. arXiv:1307.0031 [physics].

[31] R. Kleinberg, “Geographic Routing Using Hyperbolic Space,” in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, (Anchorage, AK, USA), pp. 1902–1909, IEEE, 2007.

[32] M. Nickel and D. Kiela, “Poincaré embeddings for learning hierarchical representations,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS’17, (Red Hook, NY, USA), pp. 6341–6350, Curran Associates Inc., Dec. 2017.

[33] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric Deep Learning: Going beyond Euclidean data,” *IEEE Signal Processing Magazine*, vol. 34, pp. 18–42, July 2017. Conference Name: IEEE Signal Processing Magazine.

[34] M. Nickel and D. Kiela, “Learning Continuous Hierarchies in the Lorentz Model of Hyperbolic Geometry,” in *Proceedings of the 35th International Conference on Machine Learning*, pp. 3779–3788, PMLR, July 2018. ISSN: 2640-3498.

[35] R. Shimizu, Y. Mukuta, and T. Harada, “Hyperbolic Neural Networks++,” Mar. 2021. arXiv:2006.08210 [cs, stat].

[36] E. Mathieu, C. Le Lan, C. J. Maddison, R. Tomioka, and Y. W. Teh, “Continuous Hierarchical Representations with Poincaré Variational Auto-Encoders,” in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[37] P. Kolyvakis, A. Kalousis, and D. Kiritsis, “Hyperbolic Knowledge Graph Embeddings for Knowledge Base Completion,” in *The Semantic Web* (A. Harth, S. Kirrane, A.-C. Ngonga Ngomo, H. Paulheim, A. Rula, A. L. Gentile, P. Haase, and M. Cochez, eds.), Lecture Notes in Computer Science, (Cham), pp. 199–214, Springer International Publishing, 2020.

[38] I. Chami, A. Wolf, D.-C. Juan, F. Sala, S. Ravi, and C. Ré, "Low-Dimensional Hyperbolic Knowledge Graph Embeddings," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, (Online), pp. 6901–6914, Association for Computational Linguistics, July 2020.

[39] Y. Chen, M. Yang, Y. Zhang, M. Zhao, Z. Meng, J. Hao, and I. King, "Modeling Scale-free Graphs with Hyperbolic Geometry for Knowledge-aware Recommendation," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, (New York, NY, USA), pp. 94–102, Association for Computing Machinery, Feb. 2022.

[40] L. Sun, Z. Zhang, J. Zhang, F. Wang, H. Peng, S. Su, and P. S. Yu, "Hyperbolic Variational Graph Neural Network for Modeling Dynamic Graphs," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 4375–4383, May 2021. Number: 5.

[41] S. Yan, Y. Xiong, and D. Lin, "Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, Apr. 2018. Number: 1.

[42] Y. Zhang, X. Wang, C. Shi, N. Liu, and G. Song, "Lorentzian Graph Convolutional Networks," in *Proceedings of the Web Conference 2021*, WWW '21, (New York, NY, USA), pp. 1249–1261, Association for Computing Machinery, June 2021.

[43] S. Zhu, S. Pan, C. Zhou, J. Wu, Y. Cao, and B. Wang, "Graph Geometry Interaction Learning," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 7548–7558, Curran Associates, Inc., 2020.

[44] A. Gu, F. Sala, B. Gunel, and C. Ré, "Learning Mixed-Curvature Representations in Product Spaces," Sept. 2018.

[45] M. Yang, M. Zhou, L. Pan, and I. King, "HGCN: Tree-likeness Modeling via Continuous and Discrete Curvature Learning," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, (New York, NY, USA), pp. 2965–2977, Association for Computing Machinery, Aug. 2023.

[46] X. Fu, J. Li, J. Wu, Q. Sun, C. Ji, S. Wang, J. Tan, H. Peng, and P. S. Yu, "ACE-HGNN: Adaptive Curvature Exploration Hyperbolic Graph Neural Network," in *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 111–120, Dec. 2021. ISSN: 2374-8486.

[47] M. Law, "Ultrahyperbolic Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 22058–22069, Curran Associates, Inc., 2021.

[48] Z. Xie, X. Zuo, and Y. Song, "Trivial bundle embeddings for learning graph representations," Dec. 2021. arXiv:2112.02531 [cs].

[49] T. Yu and C. De Sa, "Random Laplacian Features for Learning with Hyperbolic Space," Mar. 2023. arXiv:2202.06854 [cs] version: 2.

[50] N. Choudhary, N. Rao, K. Subbian, and C. Reddy, *Text Enriched Sparse Hyperbolic Graph Convolutional Networks.* July 2022.

[51] F. Wei, M. Ping, and K. Mei, *Adaptive Frequency-Based Fully Hyperbolic Graph Neural Networks.* June 2022.

[52] Y. Liu, B. Lang, and F. Quan, "MST-HGCN: a minimum spanning tree hyperbolic graph convolutional network," *Applied Intelligence*, vol. 53, pp. 14515–14526, June 2023.

[53] A. Tifrea, G. Bécigneul, and O.-E. Ganea, "Poincaré GloVe: Hyperbolic Word Embeddings," Nov. 2018. arXiv:1810.06546 [cs].

[54] A. Cacciola, A. Muscoloni, V. Narula, A. Calamuneri, S. Nigro, E. A. Mayer, J. S. Labus, G. Anastasi, A. Quattrone, A. Quartarone, D. Milardi, and C. V. Cannistraci, "Coalescent embedding in the hyperbolic space unsupervisedly discloses the hidden geometry of the brain," May 2017. arXiv:1705.04192 [cond-mat, q-bio].

[55] B. P. Chamberlain, S. R. Hardwick, D. R. Wardrope, F. Dzogang, F. Daolio, and S. Vargas, "Scalable Hyperbolic Recommender Systems," Feb. 2019. arXiv:1902.08648 [cs, stat].

[56] X. Fu, Y. Wei, Q. Sun, H. Yuan, J. Wu, H. Peng, and J. Li, "Hyperbolic Geometric Graph Representation Learning for Hierarchy-imbalance Node Classification," in *Proceedings of the ACM Web Conference 2023*, WWW '23, (New York, NY, USA), pp. 460–468, Association for Computing Machinery, Apr. 2023.

[57] M. Kochurov, S. Ivanov, and E. Burnaev, "Are Hyperbolic Representations in Graphs Created Equal?," July 2020. arXiv:2007.07698 [cs, stat].

[58] A. Jain, "Analyzing OpenStreetMap Road Network Attributes with NetworkX, PyG and Graph Neural Networks," Jan. 2022.

[59] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, (Atlanta, GA, USA), pp. III–1058–III–1066, JMLR.org, June 2013.

[60] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social Networks*, vol. 5, pp. 109–137, June 1983.

[61] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical Review E*, vol. 78, p. 046110, Oct. 2008. Publisher: American Physical Society.

[62] T. Bläsius, T. Friedrich, M. Katzmann, U. Meyer, M. Penschuck, and C. Weyand, "Efficiently generating geometric inhomogeneous and hyperbolic random graphs," *Network Science*, vol. 10, pp. 361–380, Dec. 2022. Publisher: Cambridge University Press.

[63] P. Erdős and A. Rényi, "On the Evolution of Random Graphs," 1984.

[64] M. Leimeister and B. J. Wilson, "Skip-gram word embeddings in hyperbolic space," May 2019. arXiv:1809.01498 [cs, stat].

[65] J. W. Robbin and D. A. Salamon, *Introduction to Differential Geometry.* Berlin, Heidelberg: Springer, 2022.

A Hyperparameter configurations

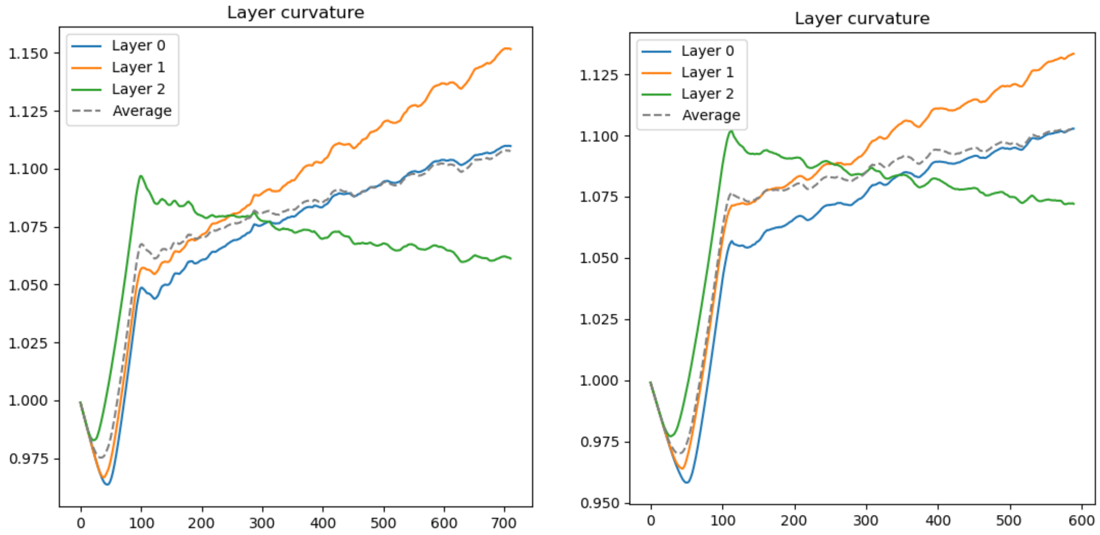
The chosen hyperparameters configurations based on the procedure proposed in section 3 are given in Table 11.

	GCN	HGCN
Cora	lr 0.01, dropout 0.2	lr 0.001, dropout 0.5, weight decay 0.001
Pubmed	lr 0.01	lr 0.01, dropout 0.5, weight decay 0.0001
Airport	lr 0.1, weight decay 0.001	lr 0.1, weight decay 0.001
Disease	lr 0.1	lr 0.1
Road	lr 0.01	lr 0.01
HRG	lr 0.1	lr 0.1
LFR	lr 0.01, dropout 0.5, weight decay 0.001	lr 0.01, dropout 0.5
SBM	lr 0.1, dropout 0.2, weight decay 0.0001	lr 0.1, weight decay 0.0001

Table 11: Hyperparameter choices for all datasets and models. Unless specified, no dropout or weight decay is used.

B Curvature training plots

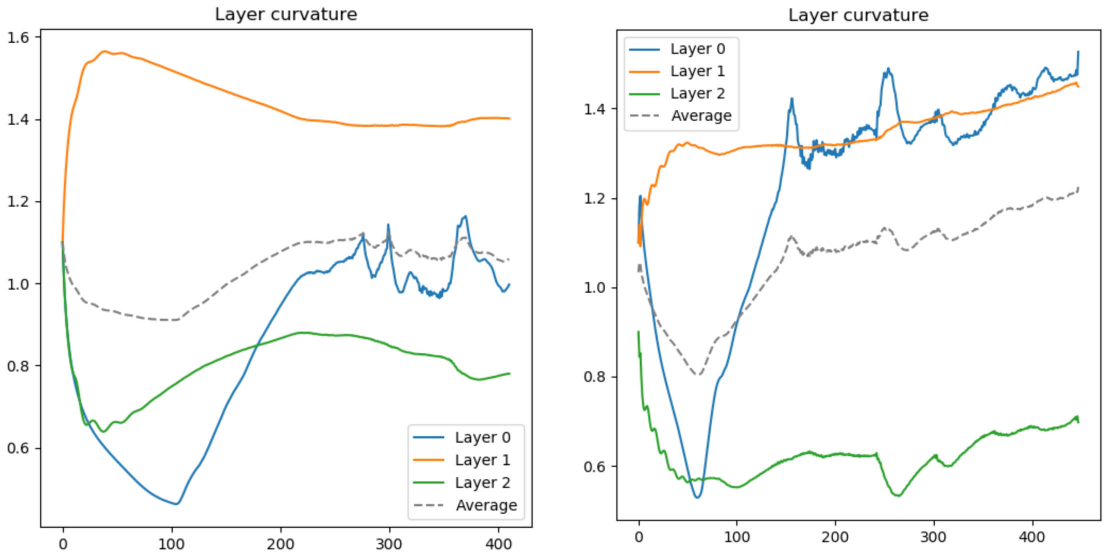
This appendix section shows the curvature training plots for each dataset (Figure 20 - Figure 26). These are achieved by training 3 times with different random seed, and picking 2 of those to plot. In each plot, the horizontal axis shows the epoch and the vertical axis shows the value of the curvature at each layer at that epoch.



(a) Run 1

(b) Run 2

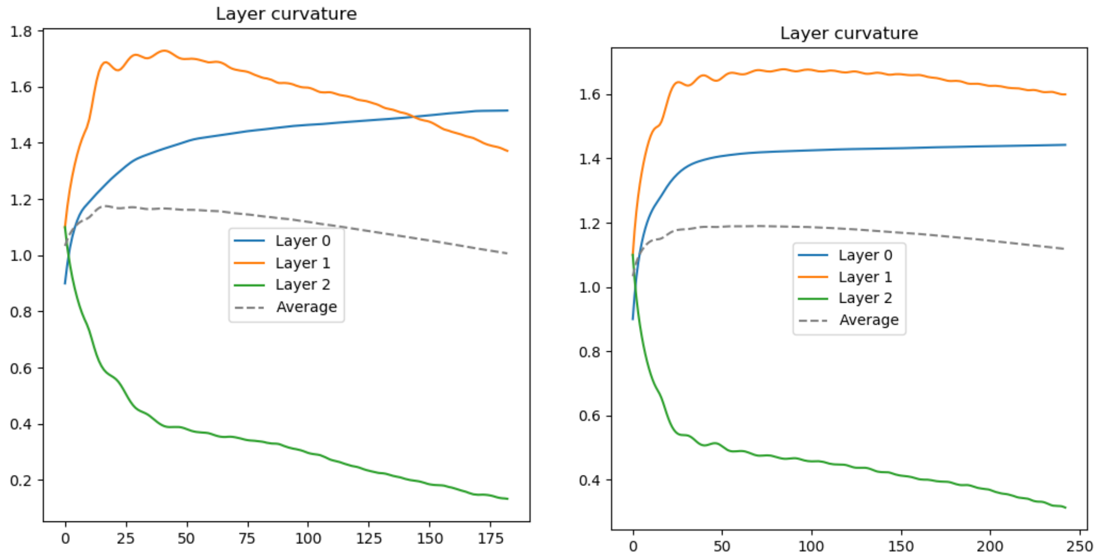
Figure 20: Learnable curvature optimization on Cora.



(a) Run 1

(b) Run 2

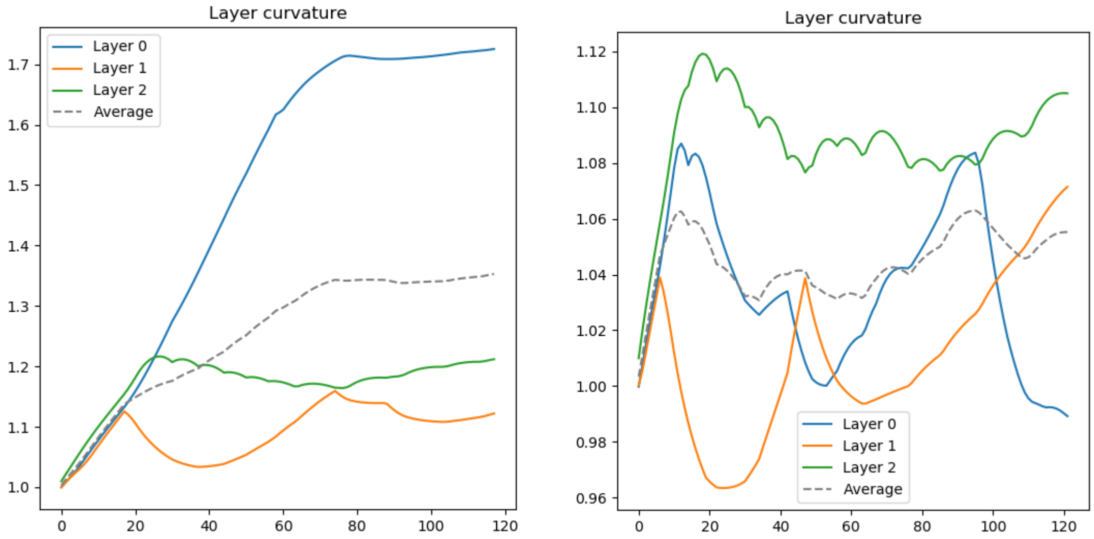
Figure 21: Learnable curvature optimization on Airport.



(a) Run 1

(b) Run 2

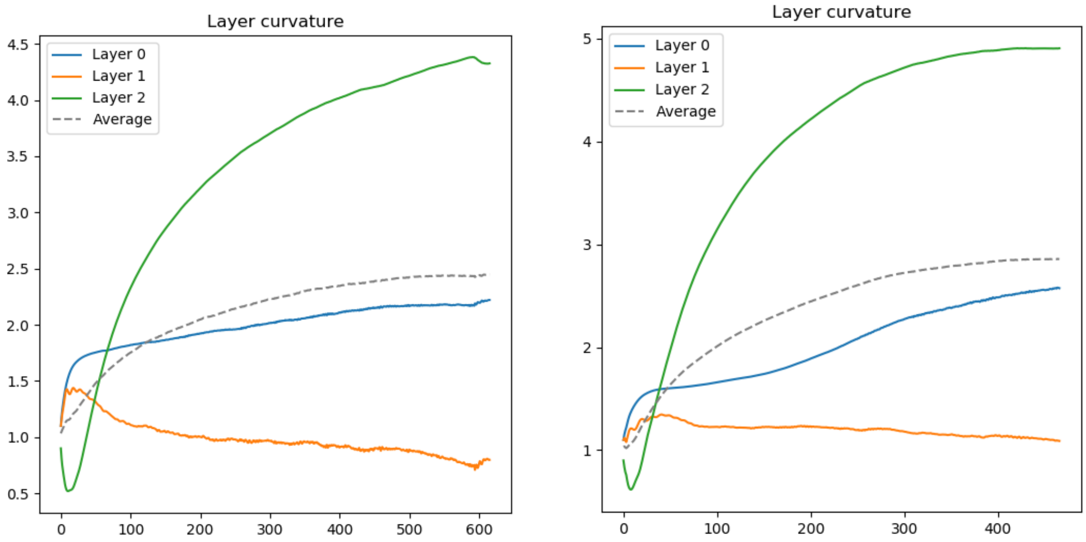
Figure 22: Learnable curvature optimization on Disease.



(a) Run 1

(b) Run 2

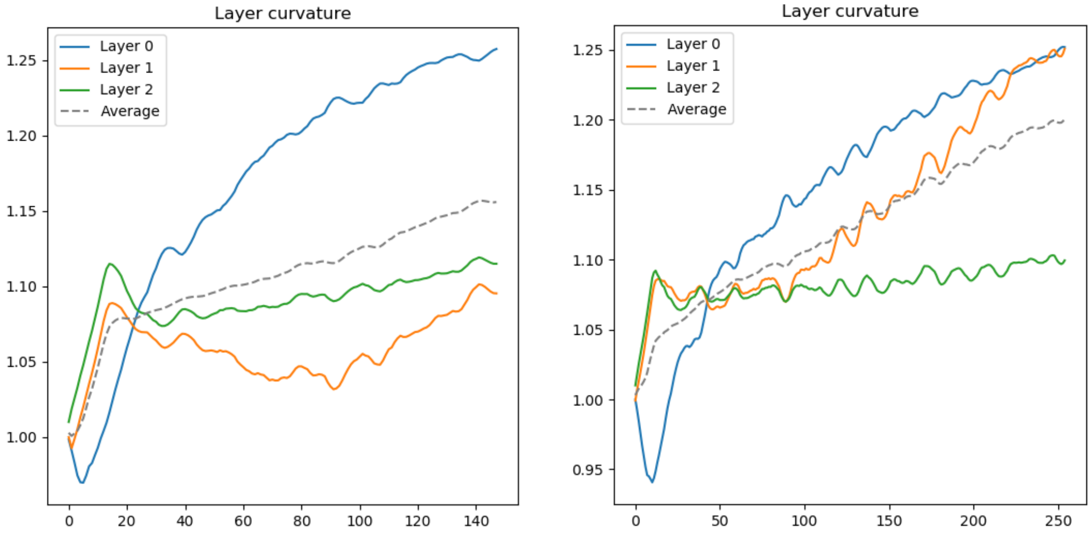
Figure 23: Learnable curvature optimization on Road.



(a) Run 1

(b) Run 2

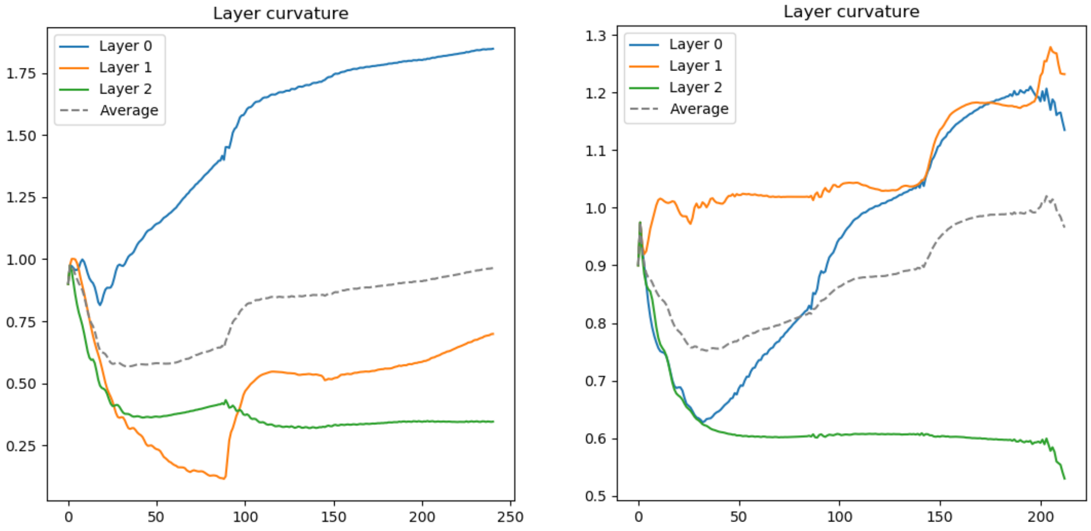
Figure 24: Learnable curvature optimization on HRG.



(a) Run 1

(b) Run 2

Figure 25: Learnable curvature optimization on LFR.



(a) Run 1

(b) Run 2

Figure 26: Learnable curvature optimization on SBM.

C Formulas for Riemannian manifolds and hyperbolic space

Here we give the relevant definitions and formulas for Riemannian spaces and hyperbolic space. Most of this is summarized from Chami et al. [1]. Some derivations and proofs that are omitted here can be found there. A Riemannian manifold is a pair $(\mathcal{M}, g)$ with a smooth manifold $\mathcal{M}$ and Riemannian metric g; $g_{\mathbf{x}} : \mathcal{T}_{\mathbf{x}}\mathcal{M} \times$

$\mathcal{T}_\mathbf{x}\mathcal{M} \to \mathbb{R}$ for $\mathbf{x} \in \mathcal{M}$. The Riemannian metric gives the manifold a measure the rate of change, and thus the distance between points. The norm induced by the Riemannian metric is denoted by $||\mathbf{v}||_g = \sqrt{g_\mathbf{x}(\mathbf{v},\mathbf{v})}$ for $\mathbf{v} \in \mathcal{T}_\mathbf{x}\mathcal{M}$.
The Riemannian metric γ allows us to directly define the length of a curve on the manifold, given by (14). Here $\gamma : [a,b] \to \mathcal{M}$ is a smooth curve on $\mathcal{M}$. $\gamma(a)$ and $\gamma(b)$ are the start- and endpoints of the curve, respectively, t is a sort of time measure, and the derivative over t measures the rate of change when moving along the curve.

$$L(\gamma) := \int_a^b ||\gamma'(t)||_g dt \tag{14}$$

We can now define distances and geodesics. The distance $d : \mathcal{M} \times \mathcal{M} \to [0,\infty)$ between two points on the manifold is given by (15). It is simply the length of the shortest curve between the two points. A geodesic is exactly this shortest curve, defined by (16), and is a generalization of straight lines to non-Euclidian space.

$$d(\mathbf{x},\mathbf{y}) := \inf_\gamma \{L(\gamma) : \gamma \text{ is a continuously differentiable curve joining } \mathbf{x} \text{ and } \mathbf{y}\} \tag{15}$$

$$\gamma : [a,b] \to \mathcal{M} \text{ is geodesic if } d(\gamma(t),\gamma(s)) = L(\gamma|_{[t,s]}) \forall (t,s) \in [a,b](t < s) \tag{16}$$

Next we explain formulas for both models of hyperbolic space. Some derivations are explained at the end.

Poincaré ball model

Poincaré ball model of dimension d, radius 1, and constant negative curvature -1 is the Riemannian manifold $(\mathbb{D}^{d,1}, (g_\mathbf{x})_\mathbf{x})$ with

$$\mathbb{D}^{d,1} := \left\{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\|_2 < 1\right\}$$

and

$$g_\mathbf{x} = \lambda_\mathbf{x}^2 I_d$$

where $|| \cdot ||_2$ is the Euclidian 2-norm and $\lambda_\mathbf{x} := \frac{2}{1-\|\mathbf{x}\|_2^2}$. Note that $g_\mathbf{x}(\mathbf{u},\mathbf{v}) := \mathbf{u} \cdot g_\mathbf{x} \cdot \mathbf{v}$ with $g_\mathbf{x}$ being a matrix that defines the product at $\mathbf{x}$.
Thus we have $||\mathbf{u}||_{g_\mathbf{x}} = \sqrt{<\mathbf{u},\mathbf{u}>_{g_\mathbf{x}}} = \frac{2||\mathbf{u}||_2^2}{1-||\mathbf{x}||_2^2}$, meaning the norm of the tangent goes to infinity as $\mathbf{x}$ goes away from the origin. Note that since the disk is in $\mathbb{R}^2$, all the tangent spaces are also just $\mathbb{R}^2$. With this norm we can find an expression for the distance between to points $\mathbf{x}$ and $\mathbf{y}$ on the disk:

$$d_\mathbb{D}^1(\mathbf{x},\mathbf{y}) = \operatorname{arcosh}\left(1 + 2\frac{\|\mathbf{x}-\mathbf{y}\|_2^2}{\left(1-\|\mathbf{x}\|_2^2\right)\left(1-\|\mathbf{y}\|_2^2\right)}\right)$$

Other formulas for the Poincaré disk are omitted but can be derived in a similar way as those for the Hyperboloid model.

Hyperboloid model

The Hyperboloid model is based on the Minkowski inner product. The Minkowski inner product $\langle .,. \rangle_\mathcal{L} : \mathbb{R}^{d+1} \times \mathbb{R}^{d+1} \to \mathbb{R}$ is given by:

$$\langle \mathbf{x},\mathbf{y} \rangle_\mathcal{L} := -x_0y_0 + x_1y_1 + \ldots + x_dy_d$$

The hyperboloid model with unit imaginary radius and constant negative curvature $-\frac{1}{K}$ in d dimensions is defined as the Riemannian manifold $(\mathbb{H}^{d,1}, (g_\mathbf{x})_\mathbf{x})$ where

$$\mathbb{H}^{d,K} := \{\mathbf{x} \in \mathbb{R}^{d+1} : \langle \mathbf{x},\mathbf{x} \rangle_\mathcal{L} = -K, x_0 > 0\}$$

and

$$g_{\mathbf{x}} := \begin{bmatrix} -1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}.$$

Tangent spaces: The tangent spaces of the hyperboloid model are given by

$$\mathcal{T}_{\mathbf{x}}\mathbb{H}^{d,K} := \left\{\mathbf{v} \in \mathbb{R}^{d+1} : \langle \mathbf{v}, \mathbf{x} \rangle_{\mathcal{L}} = 0\right\}.$$

This follows from the definitions of the hyperboloid model and the tangent space.
Distance: The distance between two points in the hyperboloid is given by

$$d_{\mathcal{L}}^{K}(\mathbf{x}, \mathbf{y}) = \sqrt{K}\,\text{arcosh}(-\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}/K)$$

Geodesics: Let $\mathbf{x} \in \mathbb{H}^{d,K}$ and $\mathbf{u} \in \mathcal{T}_{\mathbf{x}}\mathbb{H}^{d,1}$ with $\mathbf{u}$ unit speed ($\langle \mathbf{u}, \mathbf{u} \rangle_{\mathcal{L}} = 1$). The unit-speed geodesic $\gamma^{k}_{\mathbf{x}\to\mathbf{u}} :\to \mathbb{H}^{d,1}$ such that $\gamma_{\mathbf{x}\to\mathbf{u}}(0) = \mathbf{x}$ and $\dot{\gamma}_{\mathbf{x}\to\mathbf{u}}(0) = \mathbf{u}$ is given by

$$\gamma^{K}_{\mathbf{x}\to\mathbf{u}}(t) = \cosh\left(\frac{t}{\sqrt{K}}\right)\mathbf{x} + \sqrt{K}\sinh\left(\frac{1}{\sqrt{K}}\right)\mathbf{u}.$$

This is equivalent to a straight line passing through $\mathbf{x}$ in direction $\mathbf{u}$.
Parallel transport: If two points $\mathbf{x}$ and $\mathbf{y}$ on the hyperboloid $\mathbb{H}^{d,1}$ are connected by a geodesic, then the parallel transport of a tangent vector $\mathcal{T}_{\mathbf{x}}\mathbb{H}^{d,1}$ to the tangent space $\mathcal{T}_{\mathbf{y}}\mathbb{H}^{d,1}$ is:

$$P_{\mathbf{x}\to\mathbf{y}}(\mathbf{v}) = \mathbf{v} - \frac{\langle \log_{\mathbf{x}}(\mathbf{y}), \mathbf{v} \rangle_{\mathcal{L}}}{d^{1}_{\mathcal{L}}(\mathbf{x}, \mathbf{y})^2}\left(\log_{\mathbf{x}}(\mathbf{y}) + \log_{\mathbf{y}}(\mathbf{x})\right)$$

Projections: A point $\mathbf{x} = (x_0, \mathbf{x}_{1:d}) \in \mathbb{R}^{d+1}$ can be projected on $\mathbb{H}^{d,1}$ with:

$$\Pi_{\mathbb{R}^{d+1}\to\mathbb{H}^{d,1}}(\mathbf{x}) := \left(\sqrt{1 + \|\mathbf{x}_{1:d}\|_2^2}, \mathbf{x}_{1:d}\right).$$

Similarly, a point $\mathbf{v} \in \mathbb{R}^{d+1}$ can be projected on $\mathcal{T}_{\mathbf{x}}\mathbb{H}^{d,1}$ with:

$$\Pi_{\mathbb{R}^{d+1}\to\mathcal{T}_{\mathbf{x}}\mathbb{H}^{d,1}}(\mathbf{v}) := \mathbf{v} + \langle \mathbf{x}, \mathbf{v} \rangle_{\mathcal{L}}\mathbf{x}.$$

These projections can be used to make sure that certain points remain in the manifold or the tangent space after certain operations in the implementation.
Exponential and logarithmic maps: For $\mathbf{x} \in \mathbb{H}^{d,K}, \mathbf{v} \in \mathcal{T}_{\mathbf{x}}\mathbb{H}^{d,K}$ and $\mathbf{y} \in \mathbb{H}^{d,K}$ such that $\mathbf{v} \neq \mathbf{0}$ and $\mathbf{y} \neq \mathbf{x}$, the exponential and logarithmic maps of the hyperboloid model are given by:

$$\exp^{K}_{\mathbf{x}}(\mathbf{v}) = \cosh\left(\frac{\|\mathbf{v}\|_{\mathcal{L}}}{\sqrt{K}}\right)\mathbf{x} + \sqrt{K}\sinh\left(\frac{\|\mathbf{v}\|_{\mathcal{L}}}{\sqrt{K}}\right)\frac{\mathbf{v}}{\|\mathbf{v}\|_{\mathcal{L}}}$$

$$\log^{K}_{\mathbf{x}}(\mathbf{y}) = d^{K}_{\mathcal{L}}(\mathbf{x}, \mathbf{y})\frac{\mathbf{y} + \frac{1}{K}\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}\mathbf{x}}{\left\|\mathbf{y} + \frac{1}{K}\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}\mathbf{x}\right\|_{\mathcal{L}}}.$$

The exponential map maps a vector from the tangent space of a point to the hyperboloid. The logarithmic map maps a point from the hyperboloid to a vector in the tangent space of another point. These can be used to go back and forth from the manifold to a tangent space, where certain operations can be done.

Projection between Poincaré ball and hyperboloid model

The following equations show how to project points between the two models, which can be useful for example for visualizing a point on the Poincaré disk after computing it in the Hyperboloid model.

$$\Pi_{\mathbb{H}^{d,1}\to\mathbb{D}^{d,1}}(x_0, \ldots, x_d) = \frac{(x_1, \ldots, x_d)}{x_0 + 1}$$

$$\Pi_{\mathbb{D}^{d,1}\to\mathbb{H}^{d,1}}(x_1, \ldots, x_d) = \frac{\left(1 + \|\mathbf{x}\|_2^2, 2x_1, \ldots, 2x_d\right)}{1 - \|\mathbf{x}\|_2^2}$$

Derivations of formulas for Hyperboloid model

Here we give a brief overview on how one would go about deriving the formulas of the Hyperboloid model, meant to give the reader a better understanding of the model and its operations.
Before looking at the formulas it is useful to look at the definitions of the hyperbolic functions in Figure 27. What is important is that cosh and sinh are defined as parametrizations of the hyperboloid. We can use this to define the geodesics on the hyperboloid.

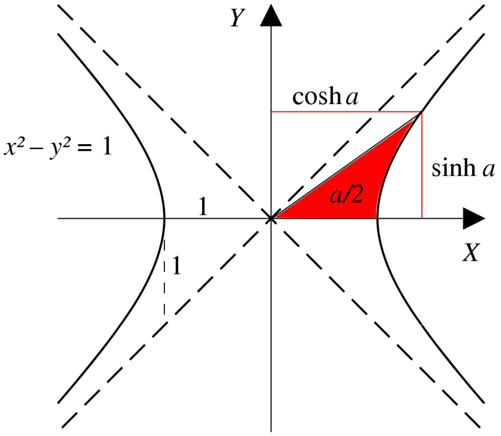


Figure 27: Hyperbolic functions; related formula is $\cosh^2 x - \sinh^2 x = 1$

Geodesics: The geodesic crossing point $\mathbf{x}$ on the hyperboloid in the direction $\mathbf{u}$ must satisfy the following conditions: $\mathbf{x}$ is on the hyperboloid and $\mathbf{u}$ is a unit tangent vector so $\langle \mathbf{x}, \mathbf{x} \rangle_{\mathcal{L}} = -1$; the norm $\langle \mathbf{u}, \mathbf{u} \rangle_{\mathcal{L}} = 1$; the product of $\langle \mathbf{x}, \mathbf{u} \rangle_{\mathcal{L}} = 0$; the geodesic lies in the plane spanning $\mathbf{x}$ and $\mathbf{u}$ so $c(t) = a(t)\mathbf{x} + b(t)\mathbf{u}$; the geodesic lies on the hyperboloid so $\langle c(t), c(t) \rangle_{\mathcal{L}} = -1$; the geodesic has unit speed so $\langle c'(t), c'(t) \rangle_{\mathcal{L}} = 1$; the geodesic starts at $\mathbf{x}$ with velocity $\mathbf{u}$ so $c(0) = \mathbf{x}$ and $c'(0) = \mathbf{u}$.
The unique solution to these conditions is $a(t) = \cosh(t), b(t) = \sinh(t)$. We do not give a proof but it can be seen as the same process as parametrizing a geodesic on a sphere, which would be $(t) = \cos(t)\mathbf{x} + \sin(t)\mathbf{u}$ based on the identity $\cos^2(t) + \sin^2(t) = 1$, whereas in the hyperbolic case this is $\cosh^2(t) - \sinh^2(t) = 1$.
Exponential and logarithmic maps: The exponential and logarithmic maps follow from the geodesic definition and are explained by Chami et al. [1] in an appendix. The idea that a vector in the tangent space is a projection of a geodesic going through that point.
Parallel transport: For the parallel transport formula the proof includes taking the covariant derivative along a geodesic. The proof is given in an appendix of [64] based on theory from [65].
Projections: The projections to the hyperboloid and the tangent space are given in [65]. They are motivated by the geometric structure, and so are the projections between the hyperboloid and the Poincaré disk.
Distance: The distance formula is derived directly from the geodesic formula. Consider $\mathbf{x}$ and $\mathbf{y}$ on the hyperboloid. We know there is a $\mathbf{u} \in \mathcal{T}_{\mathbf{x}}\mathbb{H}$ that defines the geodesic between $\mathbf{x}$ and $\mathbf{y}$. We know that $\langle \mathbf{x}, \mathbf{x} \rangle_{\mathcal{L}} = -1$, $\langle \mathbf{x}, \mathbf{u} \rangle_{\mathcal{L}} = 0$, and $\mathbf{y} = \mathbf{x}\cosh(t) + \mathbf{u}\sinh(t)$ with $t > 0$ and $t = d(\mathbf{x}, \mathbf{y})$. Then we simply derive the distance:

$$\begin{aligned}
\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} &= \langle \mathbf{x}, \mathbf{x}\cosh(t) + \mathbf{u}\sinh(t) \rangle_{\mathcal{L}} \\
\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} &= \langle \mathbf{x}, \mathbf{x} \rangle_{\mathcal{L}} \cosh(t) + 2\langle \mathbf{x}, \mathbf{u} \rangle_{\mathcal{L}} \cosh(t)\sinh(t) + \langle \mathbf{x}, \mathbf{u} \rangle_{\mathcal{L}} \sinh(t) \\
\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} &= 1 \cdot \cosh(t) + 0 \cdot \cosh(t)\sinh(t) + 0 \cdot \sinh(t) \\
\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} &= \cosh(t) \\
t &= \operatorname{arcosh}(-\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}) \\
d(\mathbf{x}, \mathbf{y}) &= \operatorname{arcosh}(-\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}})
\end{aligned}$$