A Unification Framework for Euclidean and Hyperbolic Graph Neural Networks

Anonymous submission

Abstract

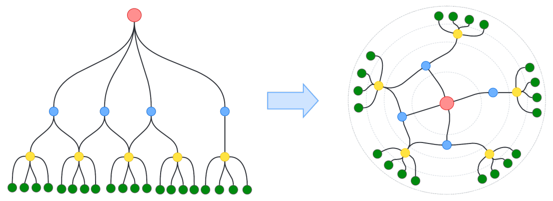
Hyperbolic neural networks are able to capture the inherent hierarchy of graph datasets, and consequently a powerful choice of GNNs. However, they entangle multiple incongruent (gyro-)vector spaces within a layer, which makes them limited in terms of generalization and scalability. In this work, we propose to use Poincaré disk model as our search space, and apply all approximations on the disk (as if the disk is a tangent space derived from the origin), and thus getting rid of all inter-space transformations. Such an approach enables us to propose a hyperbolic normalization layer, and to further simplify the entire hyperbolic model to a Euclidean model cascaded with our hyperbolic normalization layer.

We applied our proposed nonlinear hyperbolic normalization to the current state-of-the-art homogeneous and multi-relational graph networks. We demonstrate that not only does the model leverage the power of Euclidean networks such as interpretability and efficient execution of various model components, but also it outperforms both Euclidean and hyperbolic counterparts in our benchmarks.

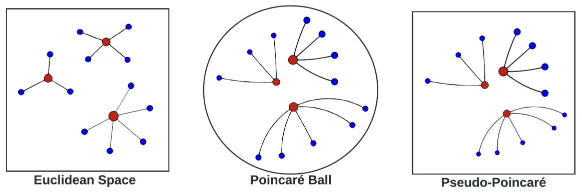
Introduction

Graph data structures are ubiquitous. There have been several advancements in graph processing techniques across several domains such as citation networks [24], medicine [1], and e-commerce [7]. Recently, non-Euclidean geometries such as hyperbolic spaces have shown significant potential in representation learning and graph prediction [6]. Due to their exponential growth in volume with respect to radius [15], hyperbolic spaces are able to embed hierarchical structures with low distortion, i.e., increase in the number of nodes in a tree with increasing depth is analogous to moving outwards from the origin in a Poincaré ball (since both grow exponentially). This similarity enables the hyperbolic space to project the child nodes exponentially further away from the origin compared to their parents (as shown in Figure 1.a) which captures hierarchy in the embedding space. However, there are certain fundamental questions about hyperbolic models which are yet to be studied. For example;

- Hyperbolic models operate on completely new manifolds with their own gyrovector space [29] (to be differentiated from the Euclidean vector space). Because they operate in different manifold, techniques such as dropout or L1/2 regularization does not guarantee similar behavior.



(a)



(b)

Figure 1: (a) Mapping hierarchical data onto a Poincaré ball. Starting with the root as origin (red), the children are continuously embedded further away with exponentially increasing distances. (b) Aggregation of vectors in Euclidean, Poincaré Ball, and pseudo-Poincaré space (hyperbolic vectors in Euclidean space). Red points are the aggregation of blue points. In the pseudo-Poincaré paradigm, we reformulate hyperbolic operations to capture hierarchy in the Euclidean space.

- While existing hyperbolic operations [10] provide a fair support for basic sequential neural networks, a correct translation of more complex Euclidean architectures (such as multi-headed attention networks [30]) is non-trivial and counter-intuitive, due to the special properties of Riemannian manifold [25].
- Hyperbolic operators are computationally very expensive i.e., the number of complex mathematical operations for each primitive operator (see Section 3) limits their scalability compared to the Euclidean graph neural networks.

Traditionally, hyperbolic models use a mixture of hyperbolic geodesics, gyro-vector space mathematics (e.g. Poincaré disc model combined with Möbius operations), and tangent space approximations at various points. Instead of switching between multiple frameworks with potentially incongruent operations, in this work, we let the Poincaré disk model be our search space and propose to apply all approximations inplace on the Poincaré disk (i.e. to apply all tangent space approximations on the disk as if the disk is a tangent space derived from the origin). This enables us to replace non-scalable gyro-vector/Möbius operations with a Euclidean approximation, and then simplify the whole hyperbolic model as a Euclidean model cascaded with a hyperbolic normalization function. In our approach, we replace all Möbius operations with Euclidean operations, yet we still work in Riemannian manifold (by using our proposed hyperbolic normalization layer along with using Riemannian optimizer), thus, we call it **Pseudo-Poincaré** framework (shown in Figure 1.b).

With our formulations, the trainable parameters lie in the Euclidean vector space, hence we can re-use the architectures that already exist for Euclidean solution space. This leads to the use of inexpensive operators and existing hardware infrastructures. It also enables the use of existing well-optimized Euclidean models along with their hyperparameter settings tuned for specific tasks. Having the aforementioned two capabilities, our framework brings the best of two worlds together.

For further evidence, we have applied our formulation to construct new pseudo-Poincaré variants of Graph Convolution Network GCN, Graph Attention Network GAT, and multi-relational graph embedding model MuR (we call our variants as $NGCN$, $NGAT$ and $NMuR$ respectively). We demonstrate that the reformulated variants outperform several state-of-the-art Euclidean and non-Euclidean graph networks on standard graph tasks. Furthermore, our experiments indicate our pseudo-Poincaré variants benefits from using regularization methods used in Euclidean counterparts (unlike the traditional hyperbolic counterparts). We also show the execution speed of our proposed model to be comparable to that of the Euclidean counterparts, and hence provides insights for building more complex/deeper hyperbolic (pseudo-Poincaré) models.

Related Work

Research into graph representation learning can be broadly classified into two categories based on the nature of the underlying graph dataset: (i) Homogeneous graphs - where the edges connect the nodes but do not have any underlying information and (ii) Heterogeneous or multi-relational graphs - where the edges contain information regarding the type of connection or a complex relation definition. Early research in this area relied on capturing information from the adjacency matrix through factorization and random walks. In matrix factorization based approaches [5], the adjacency matrix A is factorized into low-dimensional dense matrices L such that $\|L^T L - A\|$ is minimized. In random walk based approaches [11, 18], the nodes' information is aggregated through message passing over its neighbors through random or metapath traversal [9]. More recently, the focus has shifted towards neighborhood aggregation through neural networks, specifically, Graph Neural Networks (GNNs) [22]. In this line of work, several architectures such as GraphSage [13], GCN [14], and GAT [31] have utilized popular deep learning frameworks to aggregate information from the nodes' neighborhood. Concurrently multi-relational graph neural networks have been developed focusing on representation learning in knowledge graphs in which the nodes/entities can have edges/relations of different types. TransE [4], DistMult [32], RotatE [26], MuRE [2], and R-GCN [23] are examples of multi-relational GNNs.

Hyperbolic Networks

In order to better capture hierarchy in graph datasets, Ganea, et al. [10] proposed to use hyperbolic manifold (a form of Riemannian manifold) instead of Euclidean manifold. That work led to several architectures for representation learning in both homogeneous (HGCN [6], HAT [34]) and heterogeneous (MuRP [2], HypE [7]) graph datasets. These architectures primarily utilize the gyrovector space model to formulate the neural network operations in a Poincaré ball of curvature c as Möbius addition ($\oplus_c$), exponential map ($\exp_x^c$), logarithmic map ($\log_x^c$), Möbius scalar multiplication ($\otimes_c$), and hyperbolic activation (σ_c) [10].

While Hyperbolic Networks show promising results, switching back and forth between Möbius gyro-vector operations and Euclidean tangent approximations (at various points) makes them computationally very expensive, limits their extensibility, and makes them less predictable when applying L1/2 regularization and dropout. Furthermore, such back and forth between the manifolds (at variable points) comes with an implicit assumption that the embeddings have spatial locality in all dimensions for the models to be performant (we will later argue that this assumption is not always held).

Hyperbolic Transformation

Hyperbolic space is a Riemannian manifold with constant negative curvature. The coordinates in the hyperbolic space can be represented in several isometric models such as Beltrami-Klein Model $\mathbb{K}_c^n$ and Poincaré Ball $\mathbb{B}_c^n$. In this paper, we consider the Poincaré Ball model because it is the most widely used one for hyperbolic networks. Since most of the raw data (and true outputs) are assumed to be in the Euclidean space, the first step to use hyperbolic models is to have a mapping between the Euclidean and hyperbolic space. For an embedding $x \in \mathbb{R}^n$, its corresponding hyperbolic embedding in the Poincaré ball of curvature c, $p \in \mathbb{B}_c^n$ is calculated with the exponential map as $p = \exp_0^c(x) = \frac{tanh(\sqrt{c}|x|)}{\sqrt{c}|x|}x$. Conversely, logarithmic map is used to transform a hyperbolic embedding $p \in \mathbb{B}_c^n$ to a Euclidean vector $x \in \mathbb{R}^n$ is formulated as $x = \log_0^c(p) = \frac{tanh^{-1}(\sqrt{c}|p|)}{\sqrt{c}|p|}p$.

Hyperbolic Layers

Hyperbolic Feed-forward layer. For the Euclidean function $f : \mathbb{R}^n \to \mathbb{R}^m$, its equivalent Möbius version $f^{\otimes_c} : \mathbb{B}_c^n \to \mathbb{B}_c^m$ for the Poincaré ball is defined as $f^{\otimes_c}(p) := \exp_0^c\left(f\left(\log_0^c(p)\right)\right)$. Extending this, the hyperbolic equivalent $h^{\mathbb{B}}(p)$ of the Euclidean linear layer $h^{\mathbb{R}}(x) = Wx$ with

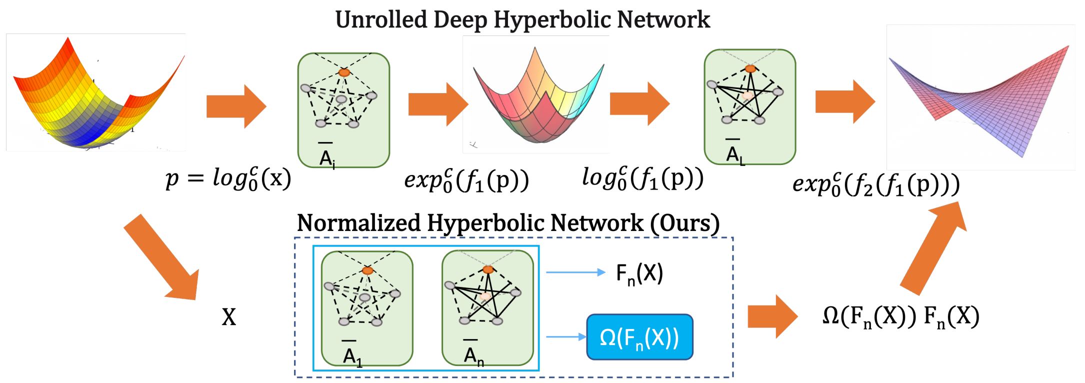


Figure 2: The top row labeled “Unrolled Deep Hyperbolic Network” presents the general idea of stacking deep hyperbolic network with L layers. The input to each layer is passed through a logarithmic map to project onto a Euclidean space, and following the Euclidean layer computation, the output is projected back onto the hyperbolic space. The section **“Hyperbolic Layers”** demonstrates how such an operator or function chaining is performed. Our methodology shown in the lowest layer completely avoids the repeated application of these subspace transformations.

weight matrix $W \in \mathbb{R}^{n\times m}$ is formulated as:

$$h^{\mathbb{B}}(p) = \frac{tanh\left(\frac{|Wp|}{|p|}tanh^{-1}\left(\sqrt{c}|p|\right)\right)}{\sqrt{c}|Wp|}Wp \qquad (1)$$

Hyperbolic Convolution. For a hyperbolic graph node representation $p_0 \in \mathbb{B}_c^n$ with neighbors $\{p_i\}_{i=1}^k$. As given in [6], the hyperbolic convolution layer $GCN^{\otimes_c}: \mathbb{B}_c^n \rightarrow \mathbb{B}_c^m$ constitutes of a linear transform, followed by neighborhood aggregation[1] which is computed as:

$$\begin{aligned} h^{\mathbb{B}}(p_i) &= W_i \otimes_c p_i \oplus_c b_i \\ h^{\mathbb{B}}_{agg}(p_0) &= exp^c_{p_0}\left(\Sigma_{i=0}^k w_{ij}\log^c_{p_0}(h^{\mathbb{B}}(p_i))\right) \\ GCN^{\otimes_c}(p_0) &= exp^c_0\left(\sigma(\log^c_0(h^{\mathbb{B}}_{agg}(p_0))\right) \end{aligned} \qquad (2)$$

where $w_{ij} = softmax_{i=1}^k(MLP(log_0^c(p_0)||log_0^c(p_i)))$ is an aggregation weight for neighbors.

Multi-relational Hyperbolic Representation. The multi-relational representations for knowledge graph KG are modeled using a scoring function. For a triplet $(h, r, t) \in KG$, where relation $x_r \in \mathbb{R}^n$ connects the head entity $x_h \in \mathbb{R}^n$ to the tail entity $x_t \in \mathbb{R}^n$ and $R \in \mathbb{R}^{n\times n}$ is the diagonal relation matrix, the hyperbolic equivalent $\phi^{\mathbb{B}}(p_h, p_r, p_t)$ of the Euclidean scoring $\phi(x_h, x_r, x_t) = -\|Rx_h - (x_t + x_r)\|^2$ function is computed using the hyperbolic distance $d^{\mathbb{B}}$ as:

$$d^{\mathbb{B}}(p_1, p_2) = \frac{2}{\sqrt{c}}\tanh^{-1}\left(\sqrt{c}\|-p_1 \oplus_c p_2\|\right) \qquad (3)$$

Pseudo-Poincaré Hyperbolic Networks

This section introduces our methodology for reformulating each of the aforementioned hyperbolic layers in Euclidean space. We base our arguments on (which will be discussed in details in the appendix):

1. Unless the spatial locality is guaranteed for the aggregated embeddings, approximating on tangent space at a variable point does not necessarily increase the approximation accuracy. Therefore, fixating tangent space to be at the origin can bring both simplicity to the model and performance in general cases.
2. Only tangent space at origin has one-to-one mapping with all points on the the Poincaré (hyperbolic) half-plane model. This enables us to combine the tangent space approximation with Poincaré half-plane model and apply all approximations in the half-plane. This makes the pseudo-Poincaré networks much easier to understand, develop, and optimize.

Pseudo-Poincaré Feed-forward Layer. We state that cascading hyperbolic feed forward layers (including linear layers) is equivalent to cascading their Euclidean counterparts and then applying a hyperbolic normalization to the result.

Lemma 1. *For a point in the tangent space at origin $x \in \mathcal{T}_{0_n}\mathbb{B}_c^n$, the exponential map to the hyperbolic space $\mathbb{B}_c^n$, $\exp_0^c : \mathcal{T}_{0_n}\mathbb{B}_c^n \rightarrow \mathbb{B}_c^n$ $\exp_0^c(x)$, is equivalent to the input scaled by a non-linear scalar function $\omega : \mathbb{R}^n \rightarrow \mathbb{R}^n$, i.e.,*

$$p = \exp_0^c(x) = \omega(x)x; \quad \omega(x) = \frac{tanh(\sqrt{c}|x|)}{\sqrt{c}|x|} \qquad (4)$$

where $\omega(x)$ is the hyperbolic normalization for exponential maps. This follows from equation (1).

Lemma 2. *Approximating on tangent space at origin for all stages causes the cascade of n hyperbolic feed-forward layers $F_n^{\otimes_c} = \{f_i^{\otimes_c}\}_{i=1}^n$ to be equivalent to cascading n of their Euclidean counterparts $F_n = \{f_i\}_{i=1}^n$ encapsulated in $\log_0^c$ and $\exp_0^c$ functions, i.e.,*

$$F_n(x) = f_n\left(f_{n-1}\left(\cdots f_1\left(x\right)\right)\right)$$
$$F_n^{\otimes_c}(p) = f_n^{\otimes_c}(f_{n-1}^{\otimes_c}(\cdots f_1^{\otimes_c}(p)) = exp_0^c\left(F_n(log_0^c(p))\right)$$

Proposition Lemma 2 implies that the hyperbolic transformations $\exp_0^c$ and $\log_0^c$ are only needed at the initial and final stages to capture hierarchy in Euclidean networks.

Lemma 3. *For given input features $x = log_0^c(p) \in \mathbb{R}^n$, a given hyperbolic feed forward layer can be rewritten as $f^{\otimes_c}(x) := \omega(f(x))f(x)$.*

From lemmas 2 and 3, we arrive at the following theorem for cascaded hyperbolic layers.

Theorem 1. *Fixating the tanget space to be always at origin, cascaded hyperbolic feed-forward layers $F_n^{\otimes_c}$ for Euclidean input $x \in \mathbb{R}^n$ can be reformulated as $F_n^{\otimes_c}(x) :=$*

[1]The aggregation occurs in the tangent space as concatenation is not well-defined for hyperbolic space.

$\Omega(F_n(x))F_n(x)$*, where* $\Omega(F_n(x))$ *is the hyperbolic normalization for a cascaded hyperbolic feed-forward network.*

Hyperbolic Normalization. Theorem 1 allows the cascaded hyperbolic layers to be reformulated in the Euclidean space using a hyperbolic normalization (Ω) function. Our normalization layer resembles the technique proposed in [21] to normalize the weights, which uses the equation; $w = \frac{e^s}{\|v\|}v$, where v is an n-dimensional vector used to learn the direction and s is a trainable parameter. However unlike weight normalization, the hyperbolic normalization happens on the features (and not the weights). Furthermore, hyperbolic normalization uses $tanh$ scaling function instead of e^s. However, both are similar in terms of using the feature vector magnitude as the denominator and hence remove the orthogonality of the dimensions.

The "unrolled deep hyperbolic network" (shown in Figure 2) presents the general idea of stacking multiple hyperbolic layers together. The input into each layer is passed through a logarithmic map to project into a Euclidean space and it's output is projected back onto the hyperbolic space after applying the computations. Equations (1)-(9) demonstrate how various hyperbolic layers are implemented.

Pseudo-Poincaré Graph Convolution. The individual components of the Hyperbolic Graph Convolution, as defined in Eq. (2), can be reformulated as a cascaded hyperbolic feed forward layers. Thus, we approximate both the aggregation weights and the aggregation function in $\mathcal{T}_0\mathbb{B}_c^n$ as:

$$h_{agg}^{\mathbb{B}}(p_i) = exp_0^c\left(\Sigma_{i=0}^{k} w_{ij}\log_0^c\left(h^{\mathbb{B}}(p_i)\right)\right) \quad (5)$$

With this approximation, we apply Theorem 1 to reformulate reformulate the $GCN^{\otimes_c} : \mathbb{B}_c^n \to \mathbb{B}_c^m$ layers as Normalized GCN $NGCN^{\otimes_c} : \mathbb{R}^n \to \mathbb{R}^m$ over Euclidean inputs $x_0 \in \mathbb{R}^n$ with neighbors $\{x_i\}_{i=1}^k$. This is computed as $NGCN^{\otimes_c} := \Omega(GCN(x_0))GCN(x_0)$, where $GCN(x_0)$ is the Euclidean Graph Convolution layer [14].

Note that the cascaded feed-forward functions of HGCN [6] operate on different tangent spaces, i.e., aggregation weights are calculated at the linear tangent space at origin $\mathcal{T}_0\mathbb{B}_c^n$ whereas the aggregation function as given in Eq. (5) is driven from a linear tangent space at the root node $\mathcal{T}_{p_0}\mathbb{B}_c^n$. An argument to use different tangent spaces for different function is based on the fact that Euclidean approximation is most accurate when the tangent space is close to the true resulting point. For aggregation, using tangent space at p_0 requires an implicit assumption that all aggregated hyperbolic points preserve a form of spatial locality, which implies similarity and not exactly the notion of hierarchy. However, we can still argue that hierarchical structures have a loose form of spatial locality, and hence using $log_0^c(p)$ will add some noise to the embedding compared to when $log_{p_0}^c(p)$ is used.

Pseudo-Poincaré Graph Attention Layer. Here, our goal is to project the Euclidean GAT layer onto the hyperbolic space by applying the hyperbolic normalization (e.g., Eq. (4)), then, investigate its approximation in relation to the existing hyperbolic graph attention layers as well as with a true hyperbolic graph attention mechanism.

Hyperbolic attention is formulated as a two-step process (as defined in [12]); (1) matching, which computes attention weights, and (2) the aggregation, which takes a weighted average of the values (attention coefficients as weights). Hence, the attention layer can be seen as two cascaded layers, in which the first calculates the attention weights, while the second layer aggregates the weighted vectors. Existing hyperbolic graph attention method [34] $GAT^{\otimes_c} : \mathbb{B}_c^n \to \mathbb{B}_c^m$ views the hyperbolic aggregation as a hyperbolic feed forward layer with a Euclidean weighted average as its function. Similarly $GAT : \mathbb{R}^n \to \mathbb{R}^m$ [31] also uses weighted-average for the aggregation.

Since we are using Euclidean weighted average over cascaded feed-forward layers, we can apply Theorem 1 to hyperbolic GAT and approximate it in the hyperbolic space with a hyperbolic normalization function and Euclidean GAT as $NGAT^{\otimes_c}(x) := \Omega(GAT(x))GAT(x)$.

Pseudo-Poincaré Multi-relational Representation. For Euclidean inputs $x_1, x_2 \in \mathbb{R}^n$ and L1-norm $d^{\mathbb{R}}(x_1, x_2)$, the hyperbolic formulation of the multi-relational hyperbolic distance $d^{\mathbb{B}}(x, y)$, given in Eq. (3) can alternatively be reformulated as $d^{\mathbb{B}}(x_1, x_2) = \omega(x)d^{\mathbb{R}}(x_1, x_2)$. Similarly, for Euclidean triplet $(x_h, x_r, x_t) \in \mathbb{R}^n$, the scoring function $\phi^{\mathbb{B}}(p_h, p_r, p_t)$ can also be approximated to $NMuR^{\mathbb{B}}(x_h, x_r, x_t)$ as:

$$\phi^{\mathbb{R}}(x_h, x_r, x_t) = -d^{\mathbb{R}}\left(Rx_h, x_t + x_r\right)^2 \quad (6)$$

$$NMuR^{\mathbb{B}}(x_h,x_r,x_t) = \Omega(\phi^{\mathbb{R}}(x_h, x_r, x_t))\phi^{\mathbb{R}}(x_h, x_r, x_t) \quad (7)$$

Optimization. The scaling function in Eq. (4) uses the vector norm in the denominator and hence, each dimension depends on the value of other dimensions (i.e., non-orthogonal). Due to this, optimizers that are developed based on the orthogonality assumptions of the dimensions such as ADAM are not expected to produce the most optimal weights. Thus, we choose Riemannian optimizers such as RiemannianADAM [3] for our model training. For traditional hyperbolic models, regularization techniques such as L2-regularization do not work as expected since the weights are in hyperbolic space. However, the trainable parameters in our formulation are in the Euclidean space. Hence, we apply regularization techniques that are applied to Euclidean networks.

Experimental Setup

In this section, we aim to answer following research questions (RQs) through our experiments:

RQ1: How do the pseudo-Poincaré variants, NGAT and NGCN, compare to their Euclidean and hyperbolic counterparts in terms of performance on node classification and link prediction?

RQ2: How does the pseudo-Poincaré variant, NMuR, compare to its Euclidean and hyperbolic counterpart in terms of performance on reasoning over knowledge graphs?

RQ3: How do pseudo-Poincaré variants react to the choice of optimizers, and what are the effects on their execution time? How does the hyperbolic normalization compare to its counterparts?

RQ4: What is the difference in the representations obtained through hyperbolic projection and hyperbolic normalization?

Problem Setting

For comparison of hyperbolic normalization with other methods, we conduct our experiments on the following tasks: (i) graph prediction (node classification and link prediction) and (ii) reasoning over knowledge graphs, which are described below:

Graph Prediction. Given a graph $\mathcal{G} = (V \times E)$, where $v_i \in \mathbb{R}^n, \{v_i\}_{i=1}^{|V|} \in V$ is the set of all nodes and $E \in \{0,1\}^{|V|\times|V|}$ is the Boolean adjacency matrix, where $|V|$ is the number of nodes in the graph and $e_{ij} = 1$, if link exists between node i and j, and $e_{ij} = 0$, otherwise. Each node $v_i \in V$ is also labeled with a class y_i. In the task of **node classification**, the primary aim is to estimate a predictor model P_θ with parameters θ such that for an input node v_i; $P_\theta(v_i|E) = y_i$. In the task of **link prediction**, the primary aim is to estimate a predictor model P_θ with parameters θ such that for an input node-pair v_i, v_j and an incomplete adjacency matrix $\hat{E}$; $P_\theta(v_i, v_j|\hat{E}) = e_{ij}$, where $e_{ij} \in E$ is an element in the complete adjacency matrix.

Reasoning over Knowledge Graphs. Let us say that a set of triplets constitutes a knowledge graph $\mathcal{KG} = \{(h_i, r_i, t_i)\}_{i=1}^{|\mathcal{KG}|}$, where $h_i \in \mathbb{R}^n$ and $t_i \in \mathbb{R}^n$ are the head and tail entity, respectively connected by a relation $r_i \in \mathbb{R}^n$. In the task of **reasoning** [2], our goal is to learn representations for the entities and relations using a scoring function that minimizes the distance between heads and tails using the relation as a transformation. The scoring function for Euclidean $MuRE$, hyperbolic $MuRP$ and pseudo-Poincaré $NMuR$ are presented in Eq. (6), and Eq. (7), respectively.

Datasets and Baselines

Datasets. For comparing our model with the baselines, we chose the following standard benchmark datasets; (i) **Cora** [20] contains 2708 publications with paper and author information connected by citation links and classified into 7 classes based on their research areas, (ii) **Pubmed** [24] contains medical publications pertaining to diabetes labeled into 3 classes. The dataset is collected from the Pubmed database, (iii) **Citeseer** [24] contains 3312 annotated scientific publications connected by citation links and classified into 6 classes of research areas, (iv) **WN18RR** [8] is a subset of the hierarchical WordNet relational graph that connects words with different types of semantic relations. WN18RR consists of 40,943 entities connected by 11 different semantic relations and (v) **FB15k-237** [4, 28] contains triple relations of the knowledge graph and textual mentions from the Freebase entity pairs, with all simple invertible relations are removed for better comparison. The dataset contains 14,541 entities connected by 237 relations. (vi) Protein-Protein Interaction (**PPI**) network dataset [13] contains 12M+ triples with 15 relations (preserving the original ratio of the relations in the dataset, we randomly sampled the dataset, and trained on the sampled subset).

We use Cora, Pubmed, and Citeseer for our experiments on homogeneous graph prediction, i.e., node classification and link prediction. For comparing the multi-relational networks on the task of reasoning, we utilize the FB15k-237 and WN18RR datasets. For a fair comparison of evaluation metrics, we use the same training, validation, and test splits as used in the previous methods, i.e., the splits for Cora, Pubmed, and Citeseer are as given in [6] and for FB15k-237 and WN18RR, the splits are in accordance with [2].

Baselines. For comparing the performance of our Pseudo-Poincaré models, we utilize the Euclidean and hyperbolic variants of the architecture as our baselines; (i) **Graph Convolution (GCN)** [14] aggregates the message of a node's k-hop neighborhood using a convolution filter to compute the neighbor's significance to the root, (ii) **Graph Attention (GAT)** [31] aggregates the messages from the neighborhood using learnable attention weights, (iii) **Hyperbolic Graph Convolution (HGCN)** [6] is a hyperbolic formulation of the GCN network that utilizes the hyperbolic space and consequently, Möbius operations to aggregate hierarchical features from the neighborhood, (iv) **Hyperbolic Graph Attention (HGAT)** [33] is a hyperbolic formulation of the GAT networks to capture hierarchical features in the attention weights, (v) **Multi-relational Embeddings (MuRE)** [2] transforms the head entity by a relation matrix and then learns representation by minimizing L1-norm between head and tail entities translated by the relation embedding and (vi) **Multi-relational Poincaré (MuRP)** [2] is a hyperbolic equivalent of MuRE that transforms the head entity by a relation matrix in the Poincaré ball and then learns representation by minimizing the hyperbolic distance between the head and tail entities translated by the relation embedding.

RQ1: Performance on Graph Prediction

To evaluate the performance of hyperbolic normalization with respect to Euclidean and hyperbolic alternatives, we compare the $NGCN$ and $NGAT$ models against the baselines on the tasks of node classification and link prediction. We utilize the standard evaluation metrics of accuracy and ROC-AUC for the tasks of node classification and link prediction, respectively. The results for our experiments are provided in Table 1.

From the results, in all cases (except link-prediction on Citeseer), pseudo-Poincaré variants outperform the Euclidean model. Pseudo-Poincaré variants perform better than their hyperbolic counterparts in node classification, but marginally lower in link-prediction. This is inline with our expectations as embeddings of adjacent points in link-prediction on homogeneous graphs tend to have spatial locality. However, spatial locality is less likely for node-classification (as far away nodes still can belong to the same class). It is also worth noting that Pseudo-Poincaré outperforms Euclidean in all node-classification datasets regardless of the hyperbolicity of the dataset (e.g., Cora has low hyperbolicity [6] which were previously suggested as a reason that inferior Hyperbolic networks over Euclidean counterparts). This observation suggests that Pseudo-Poincaré is more general purpose framework (i.e., it is performant in a wider variety of tasks/datasets compared to pure-Hyperbolic model). It should be noted that pseudo-Poincaré variants are consistently at least the second-best performing methods, closely following the top performer which makes it very appealing given the low-complexity, and speedup of the model

Table 1: Performance comparison results between hyperbolic normalization (ours) and the baseline methods on the graph prediction tasks of node classification and link prediction. The columns present the evaluation metrics, which are Accuracy (for node classification) and Area under ROC (ROC) (for link prediction), along with their corresponding 95% confidence intervals. The cells with the best performance are highlighted in bold and the second-best performance is marked with a box.

Repr.	**Datasets**	**Node Classification (Accuracy in %)**			**Link Prediction (ROC in %)**		
Space	**Models**	**CORA**	**Pubmed**	**Citeseer**	**Cora**	**Pubmed**	**Citeseer**
Euclidean	**GCN**	80.1±0.3	78.5±0.3	71.4±0.3	90.2±0.2	92.6±0.2	91.3±0.2
	GAT	$\boxed{82.7\pm0.2}$	79.0±0.3	71.6±0.3	89.6±0.2	92.4±0.2	**93.6±0.2**
Hyperbolic	**HGCN**	77.9±0.3	78.9±0.3	69.6±0.3	**91.4±0.2**	**95.0±0.1**	$\boxed{92.8\pm0.3}$
	HGAT	79.6±0.3	**79.2±0.3**	68.1±0.3	90.8±0.2	93.9±0.2	92.2±0.2
Pseudo-	**NGCN**	82.4±0.2	78.8±0.3	$\boxed{71.9\pm0.3}$	$\boxed{91.3\pm0.2}$	$\boxed{94.7\pm0.1}$	$\boxed{92.8\pm0.2}$
Poincaré	**NGAT**	**83.1±0.2**	$\boxed{79.1\pm0.3}$	**73.8±0.3**	90.5±0.2	93.9±0.2	$\boxed{92.8\pm0.2}$

Table 2: Performance comparison results between Pseudo-Poincaré (ours) and the baseline methods on the task of multi-relational graph reasoning. The columns present the evaluation metrics of Hits@K (H@K) (%) and Mean Reciprocal Rank (MRR) (%) along with their corresponding 95% confidence intervals. The best results are highlighted in bold.

	Datasets	**WN18RR**			**FB15K-237**			**PPI**		
Dim	**Models**	**MRR**	**H@10**	**H@3**	**MRR**	**H@10**	**H@3**	**MRR**	**H@10**	**H@3**
40	**MuRE**	40.9±0.3	49.7±0.3	44.5±0.3	29.0±0.3	46.2±0.3	31.9±0.3	2.81±0.3	9.24±0.3	4.11±0.3
	MuRP	42.5±0.3	52.2±0.3	45.9±0.3	29.8±0.3	47.4±0.3	32.8±0.3	5.55±0.3	10.21±0.3	4.43±0.3
	NMuR	**43.6±0.3**	**57.5±0.3**	**47.4±0.3**	**31.5±0.3**	**49.7±0.3**	**34.7±0.3**	**8.21±0.3**	**14.3±0.3**	**11.7±0.3**
200	**MuRE**	44±0.3	51.3±0.3	45.5±0.3	31.4±0.3	49.5±0.3	34.5±0.3	10.68±0.3	14.55±0.3	11.90±0.3
	MuRP	44.6±0.3	52.4±0.3	46.2±0.3	31.5±0.3	49.8±0.3	34.8±0.3	8.23±0.3	15.09±0.3	9.98±0.3
	NMuR	**44.7±0.3**	**57.9±0.3**	**48.1±0.3**	**32.2±0.3**	**50.6±0.3**	**35.5±0.3**	**12.68±0.3**	**15.07±0.3**	**13.82±0.3**

(see Table 5 for execution times).

RQ2: Multi-relational Reasoning

To compare the performance of hyperbolic normalization against the baselines for multi-relational graphs, we compare the $NMuR$ model against $MuRE$ and $MuRP$ on the task of reasoning. We consider the standard evaluation metrics of Hits@K and Mean Reciprocal Rank (MRR) to compare our methods on the reasoning task. Let us say that the set of results for a head-relation query (h, r) is denoted by $R_{(h,k)}$, then the metrics are computed by $MRR = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{rank(i)}$ and $Hits@K = \frac{1}{K}\sum_{k=1}^{K} e_k$, where $e_k = 1$, if $e_k \in R_{(h,r)}$ and 0, otherwise. Here $rank(i)$ is the rank of the i^{th} retrieved sample in the ground truth. To study the effect of dimensionality, we evaluate the models with two values of the embedding dimensions: $n = 40$ and $n = 200$. The results from our comparison are given in Table 2.

From the results, note that $NMuR$ outperforms the hyperbolic model $MuRP$, on an average, by $\approx$ 5% with 40 dimensions and $\approx$ 3% with 200 dimensions across various metrics and datasets. This shows that $NMuR$ is able to effectively learn better representations at lower number of dimensions. Also, note that the difference in performance by increasing the number of dimensions is $< 1\%$ for most cases, implying that NMuR is already able to capture the hierarchical space of the knowledge graphs at lower dimensions. This should limit the memory needed for reasoning without a significant loss in performance. (we also observed significant improvements when applying our hyperbolic normalization layer to Relational-GCN models [23]. We present the results in appendix due to lack of space.)

Table 3: Comparison of normalization methods. (Layer-Norm, Hyperbolic Norm, and Constant magnitude). The columns present results on Cora (CR), Pubmed (PM), and Citeseer (CS) datasets using GAT and GCN models. Note that the provided results are without any hyper-parameter tuning, thus the comparison is only valid column-wise.

Base Model		**CR**	**PM**	**CS**
GCN	**Constant**	67.9	76.7	68.3
	Layer-Norm	78.3	75.0	63.6
	Hyp-Norm	**80.9**	**78.0**	**72.1**
GAT	**Constant**	75.3	73.0	61.2
	Layer-Norm	75.6	74.9	65.3
	Hyp-Norm	**76.7**	**76.3**	**68.6**

RQ3: Model Analysis

Hyperbolic Normalization vs. Layer Normalization. In this experiment, we study the difference between our hyperbolic normalization compared to other existing normalization approaches. For this, we select Layer-norm [17] as it is the closest in terms of input/output. Also, in order to study the effect of hyperbolic scaling in our normalization layer, we added constant-norm which is computed as follows: $norm(x) = x/\|x\|$. In order to compare the effect of normalization layers, we did not use any extra hyper-parameter tuning and regularization. We set the embedding size to 64 for both GAT and GCN, and used 4 heads for GAT-models. Table 3 shows the comparison of different normalization methods on Cora, Pubmed, and Citeseer datasets, and confirms the outperformance of hyperbolic normalization. Since Layer-norm keeps the GAT model in the Euclidean

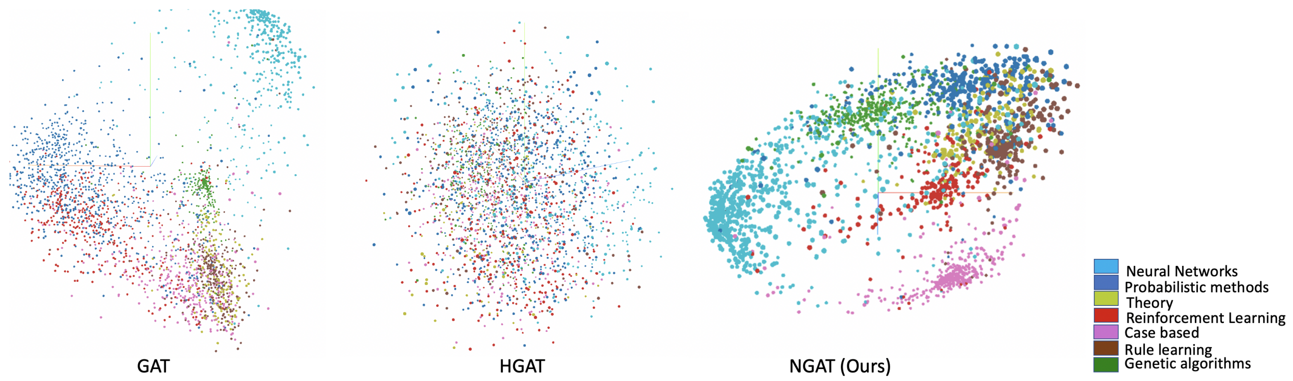


Figure 3: Embedding visualizations of the Cora citation graph for Euclidean (GAT), hyperbolic (HGAT), and our proposed Pseudo-Poincaré (NGAT) methods. NGAT achieves better separation between the node classes than the corresponding Euclidean and hyperbolic counterparts.

Table 4: Performance of NGAT, GAT, and HGAT on node classification with RAdam and Adam optimizers.

Models	Optimizers	Cora	Pubmed	Citeseer
GAT	**Adam**	**82.7±0.2**	78.7±0.2	**71.6±0.3**
	RAdam	78.4±0.3	**79.0±0.3**	71.1±0.3
HGAT	**Adam**	76.8±0.3	76.0±0.3	**68.1±0.3**
	RAdam	**79.6±0.3**	**78.9±0.3**	**68.1±0.3**
NGAT	**Adam**	78.8±0.3	77.7±0.3	69.7±0.3
	RAdam	**83.9±0.2**	**79.1±0.3**	**73.8±0.3**

Table 5: Comparison of execution times. The training epoch times (in seconds - lower is better) for Euclidean, hyperbolic, and Pseudo-Poincaré models.

Model		CR	PM	CS
Convolution	**GCN**	0.07	0.13	0.04
	HGCN	0.29	0.33	0.21
	NGCN	**0.08**	**0.14**	**0.06**
Attention	**GAT**	0.16	0.38	0.09
	HGAT	1.10	1.34	0.81
	NGAT	**0.19**	**0.37**	**0.12**

space, we used ADAM optimizer. However, since both hyperbolic normalization and constant normalization have $\|x\|$ as their denominator, we used RiemannianAdam.

Optimizer Choice. In this study, we experimentally validate our observations regarding the choice of optimizers. Table 4 shows that NGAT works better when RiemanniAdam is used as an optimizer. This shows that, although the parameters are in the Euclidean space, their gradient updates occur in accordance with hyperbolic gradients. Thus, NGAT leverages the architecture of GAT, but behaves more closely to HGAT when it comes to parameter optimization.

Execution Time Comparison. In this analysis, we demonstrate the impact of hyperbolic normalization in improving the scalability of hyperbolic networks. We study the computational cost to train our model versus the hyperbolic and Euclidean counterparts. We ran all the experiments on Quadro RTX 8000 with 48GB memory, and reported the mean of $epoch/second$ for each training experiment for link prediction task in Table 5. Our model NGCN is 3 times faster than the HGCN on average over the entire training, while NGAT is 5 times faster than its own hyperbolic counterpart (HGAT). NGCN and NGAT are 25% and 12% slower compared to their Euclidean counterparts, respectively. We observed the smallest speedup for Pubmed which is due to the very large dataset memory footprint. Even in this case, our NGCN model is still more than 2 times faster than HGCN and NGAT is more than 3.5 times faster than HGAT.

RQ4: Embedding Visualizations

We visually demonstrate that our approach of learning in Euclidean space, indeed preserves the hierarchical characteristics of hyperbolic space. Towards this goal, we present the visualization of node embeddings from the Cora citation graph. We choose Cora, since it is a citation-based graph and does not contain explicit hierarchical (is-a) relations. We contrast the embeddings learned by GAT (a method based on Euclidean space), with HGAT (learning in hyperbolic space) with the proposed NGAT approach. Figure 3 shows that NGAT ensures clearer separation between node classes in comparison to its Euclidean and hyperbolic counterparts.

Conclusion

In this paper, we showed that a Euclidean model can be converted to behave as a hyperbolic model by simply (1) connecting it to the hyperbolic normalization layer and (2) using a Riemannian optimizer. From the training perspective, this approach allows the trainable parameters to interact in the Euclidean vector space, consequently enabling the use of Euclidean regularization techniques for the model training. Furthermore, the training of our proposed approach is significantly faster compared to the existing hyperbolic methods. This allows the hyperbolic networks to be more flexible in practice to a wider variety of graph problems. From task/application perspective, we removed the implicit assumption of having spatial locality of the adjacent nodes (in all dimensions), to be a prerequisite of the hyperbolic model training. This allows the framework to be applied in a wider range of tasks.

References

[1] Anderson, R. M.; and May, R. M. 1992. *Infectious diseases of humans: dynamics and control*. Oxford university press.

[2] Balažević, I.; Allen, C.; and Hospedales, T. 2019. Multi-relational Poincaré Graph Embeddings. In *Advances in Neural Information Processing Systems*.

[3] Becigneul, G.; and Ganea, O.-E. 2019. Riemannian Adaptive Optimization Methods. In *International Conference on Learning Representations*.

[4] Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26.

[5] Cao, S.; Lu, W.; and Xu, Q. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, 891–900.

[6] Chami, I.; Ying, Z.; Ré, C.; and Leskovec, J. 2019. Hyperbolic graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, 4869–4880.

[7] Choudhary, N.; Rao, N.; Katariya, S.; Subbian, K.; and Reddy, C. K. 2021. Self-Supervised Hyperboloid Representations from Logical Queries over Knowledge Graphs. In *Proceedings of the Web Conference 2021*, WWW '21, 1373–1384.

[8] Dettmers, T.; Minervini, P.; Stenetorp, P.; and Riedel, S. 2018. Convolutional 2D Knowledge Graph Embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

[9] Fu, X.; Zhang, J.; Meng, Z.; and King, I. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *WWW*.

[10] Ganea, O.; Bécigneul, G.; and Hofmann, T. 2018. Hyperbolic neural networks. In *Advances in neural information processing systems*, 5345–5355.

[11] Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864.

[12] Gulcehre, C.; Denil, M.; Malinowski, M.; Razavi, A.; Pascanu, R.; Hermann, K. M.; Battaglia, P.; Bapst, V.; Raposo, D.; Santoro, A.; and de Freitas, N. 2019. Hyperbolic Attention Networks. In *International Conference on Learning Representations*.

[13] Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *NIPS'17*.

[14] Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

[15] Krioukov, D.; Papadopoulos, F.; Kitsak, M.; Vahdat, A.; and Boguná, M. 2010. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3): 036106.

[16] Lee, J. 2019. *Introduction to Riemannian Manifolds*. Graduate Texts in Mathematics. Springer International Publishing. ISBN 9783319917542.

[17] Lei Ba, J.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. *arXiv e-prints*, arXiv:1607.06450.

[18] Narayanan, A.; Chandramohan, M.; Venkatesan, R.; Chen, L.; Liu, Y.; and Jaiswal, S. 2017. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*.

[19] Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32: 8026–8037.

[20] Rossi, R. A.; and Ahmed, N. K. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*.

[21] Salimans, T.; and Kingma, D. P. 2016. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, 901–909. Red Hook, NY, USA: Curran Associates Inc.

[22] Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.

[23] Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Berg, R. v. d.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*, 593–607. Springer.

[24] Sen, P.; Namata, G. M.; Bilgic, M.; Getoor, L.; Gallagher, B.; and Eliassi-Rad, T. 2008. Collective Classification in Network Data. *AI Magazine*, 29(3): 93–106.

[25] Shimizu, R.; Mukuta, Y.; and Harada, T. 2021. Hyperbolic Neural Networks++. In *International Conference on Learning Representations*.

[26] Sun, Z.; Deng, Z.-H.; Nie, J.-Y.; and Tang, J. 2019. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *International Conference on Learning Representations*.

[27] Thanapalasingam, T.; van Berkel, L.; Bloem, P.; and Groth, P. 2021. Relational Graph Convolutional Networks: A Closer Look.

[28] Toutanova, K.; Chen, D.; Pantel, P.; Poon, H.; Choudhury, P.; and Gamon, M. 2015. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, 1499–1509.

[29] Ungar, A. A. 2009. *A Gyrovector Space Approach to Hyperbolic Geometry*. Synthesis Lectures on Mathematics & Statistics. Morgan & Claypool Publishers.

[30] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

[31] Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. *International Conference on Learning Representations*. Accepted as poster.

[32] Yang, B.; Yih, W.; He, X.; Gao, J.; and Deng, L. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *3rd International Conference on Learning Representations, ICLR*.

[33] Yang, T.; Hu, L.; Shi, C.; Ji, H.; Li, X.; and Nie, L. 2021. HGAT: Heterogeneous Graph Attention Networks for Semi-supervised Short Text Classification. *ACM Transactions on Information Systems (TOIS)*, 39(3): 1–29.

[34] Zhang, Y.; Wang, X.; Shi, C.; Jiang, X.; and Ye, Y. F. 2021. Hyperbolic Graph Attention Network. *IEEE Transactions on Big Data*, 1–1.

APPENDIX

Implementation Details

We run our experiments on an Nvidia T4 GPU with 16 GB of VRAM. Our models are implemented on the Pytorch framework [19] and the baselines are adopted from [6] for GAT, GCN, and HGCN, and from [2] for MuRE and MuRP. Due to the unavailability of a public implementation, we use our own implementation of HGAT based on [33]. The implementations have been thoroughly tuned with different hyper-parameter settings for the best performance.

For GAT baseline, we used 64 dimensions with 4 and 8 attention heads (we found 4 heads perform better and hence we only present that result in our tables), with dropout rate of 0.6 and L2 regularization of $\lambda = 0.0005$ (for Pubmed and Cora), and $\lambda = 0.001$ (for CiteSeer) to be inline with the hyper-parameters reported in [31]. We used same number of dimensions and hyper-parameters for our NGAT model. For GCN baseline, [31] used 64 dimensions. we used the same number of dimensions and hyperparameter setting as GAT baseline. It should be noted that, we observed poor performance when we applied regularization techniques (that GAT baseline used) to the hyperbolic models. For HGCN, the original paper used 16 hidden dimension, however, we tried 64, 32, and 16 for the hidden dimensions, we found 64 dimensions produces a better results. For the curvature choice, we used different curvatures for different tasks. For the task of node classification, we used $c = 0.3$ (for Cora), and $c = 1$ (for Pubmed and Citeseer) as they produced the best results, while for the link prediction task, $c = 1.5$ produced the best results. at the end, we empirically found that scaling the output of the hyperbolic normalization layer by the factor of 5 produces the best results when $c = 0.3 - 0.5$ (and for $c = 1.5$, we set the scaling factor to 3).

Applying Pseudo-Poincare to RGCN

In order to further show (1) the effectiveness our method, as well as (2) how easy it is to apply our method to an existing model (i.e. adoptability), we took the pytorch models implemented by [27], and added our hyperbolic normalizatoin layer to it. We also trained the model using Riemannian Adam. Table 6 shows the comparative results for the Node Classification tasks on AIFB, AM, BGS, and MUTAG datasets (further details about the datasets can be found in [27]). We used curvature $c = 1.5$, and scaling factor $s = 3$ for N-RGCN and $c = 0.5$, and $s = 5$ for NE-RGCN (except for MUTAG NE-RGCN that we used $c = 1.5$, and $s = 3$). We trained the models 10 time and picked the best checkpoint.

Table 7 shows the results Link Prediction over WN18 and FB-Toy datasets presented in 7 (we stuck to the same implementation and the datasets to have apple-to-apple comparison). We only trained RGCN (the pytorch implementation they had in 7). We used curvature $c = 1.5$, and scaling factor $s = 3$ for FB-Toy, and $c = 0.5$, and $s = 5$ for WN18 dataset. We trained the model on WN18 for 7k epochs, and on FB-toy for 12K (to replicate the experiments in [27]). We observed significant improvements in across all metrics when we applied our Hyperbolic normalization method. The fact the applying our method to the existing models is almost seemless increases the ability to explore/adopt hyperbolic models for variety of tasks/models/applications.

Table 6: Node-Classification results for RGCN model on AIFB, AM, BGS, and MUTAG datasets. Note that we used the same implementation and dataset in [27] to show that our method can be applied incrementally and effectively to the existing code-bases.
* shows the use of different hyper-parameters than the rest of the datasets.

Dataset	**AIFB**	**AM**	**BGS**	**MUTAG**
RGCN	**97.22**	89.39	82.76	67.65
N-RGCN	**97.22**	**89.9**	**89.66**	**73.53**
E-RGCN	88.89	90.4	82.76	75.00
NE-RGCN	**94.44**	**90.91**	**86.21**	**76.47***

Table 7: Link-Prediction results for RGCN model on WN18 and FB-Toy datasets. Note that we used the same implementation and dataset in [27] to show that our method can be applied incrementally and effectively to the existing code-bases.

	MPR	Hits@1	Hits3	Hits@10
Dataset	FB-Toy			
RGCN	47.0	30.9	55.3	83.2
N-RGCN	**51.8**	**35.2**	**60.9**	**86.5**
Dataset	WN18			
RGCN	65.9	43.2	89.0	93.7
N-RGCN	**75.2**	**59.8**	**90.2**	**93.9**

Fixating tangent space

We argued in the section **"Pseudo-Poincaré Hyperbolic Networks"**, we stated that fixating tangent space does not necessarily increase the approximation error where the adjacent embeddings do not show strong spatial locality property. In this appendix, we want to dive a bit more details.

Embedding aggregations is the key operation in neural networks, and is done in tangent space (in hyperbolic models). One can select one of the to-be-aggregated points as the tangent space origin. Another way is to approximate the mid-point and then derive the tangent space from that point, then to apply the approximation[2].

Since the notion of curvature becomes more complicated in high-dimensional space since the manifold can curve in many different directions, inline with the literature [16], we focus our arguments on a sub-space of the manifold (specifically, 2-D sub-space). The arguments can be generalized to higher dimensions. Given 2 points in a 2D Riemannian manifold with constant curvature (i.e., a ball or a hyperbolic-plan in the ambient space), there are infinite planes that cross-section the manifold and passes from these two points. The distance between these two points is equal to the shortest curve (which lies in the optimal plane-sections), and the

[2]this is done usually by heuristics, empirically we did not find the improvements generalizable

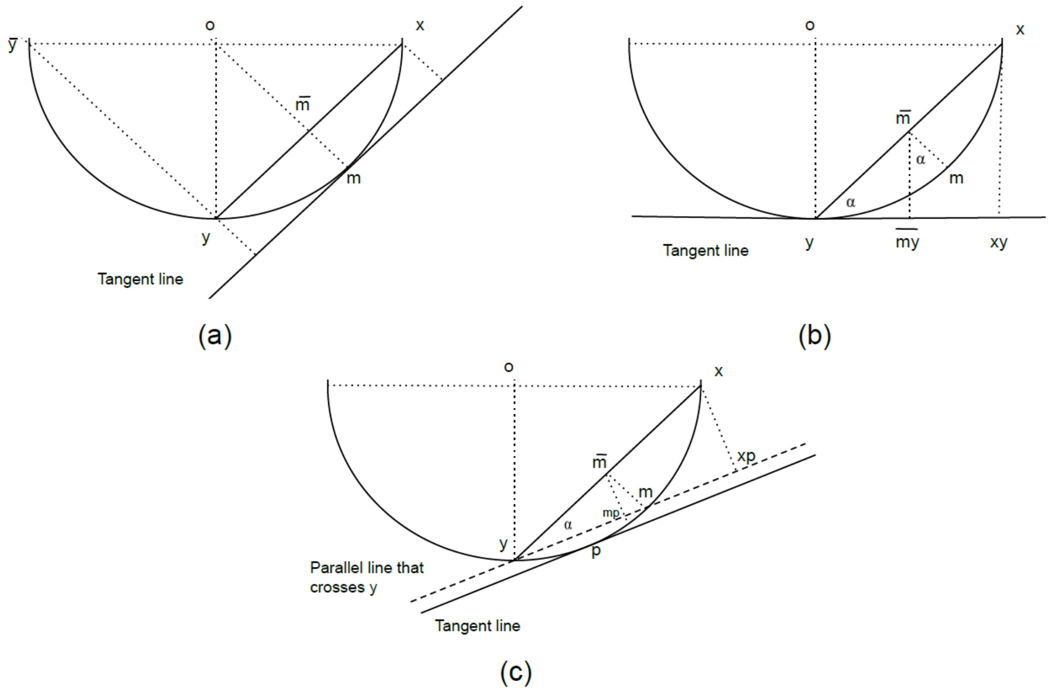


Figure 4: (a) Calculating midpoint using the ideal tangent line. (b) Calculating the midpoint using tangent line that is passing from point y. α is the angle between the tangent line (from point y) and the ideal tangent line. (c) Calculating the midpoint that uses a tangent line at arbitrary point p.

midpoint is a point with identical distance between these points.

Lemma 4. *For a tangent line from any point in the curve (with fix curvature), the midpoint approximation error is relative to the angle between the x-y chord and the tangent line.*

- The ideal tangent line is the line that is derived from the mid-point. **Explanation.** Note that curve with a contant curvature is a part of a circle, hence, the diameter from the midpoint (m) is the bisector of an xy arc. based on the **diameter-chord theorem**, it is also the bisector of the isosceles triangle formed by xy chord and half-circle radius (xoy in 4.a). This means m-diameter is perpendicular to both ideal tangent line as well as the xy chords (xy chord is parallel to the ideal tangent line). this implies that one can calculate the mid-point of the x-y line ($\overline{m}$ in 4.a) and project it to the curve (perpendicular projection) and get the ideal midpoint. Now on, we use xy chord instead of the ideal tangent line.
- Then we try to find the mid-point approximation behavior with respect to other projections. Let's start with the mid-point approximation of x,y on the curve on the tangent line derived from y (Figure 4 (b)). let us refer to the projection of x on the $\mathcal{T}_y$ (tangent line) as xy. Also let us project the $\overline{m}$ on the $\mathcal{T}_y$ and call it $\overline{mp}$. $x.x_y.y$ and $\overline{m}.\overline{m_y}.y$ are orthogonal triangles and $\overline{m}$ is mid-point of the hypotenuse xy. Based on the **mid-segment theorem**, we can infer that m_y is a mid-point of $y.x_y$, and hence its perpendicular projection on the curve gives us the approximate mid-point on the curve. Also, based on the properties of the orthogonal triangles, we can infer the ideal projection line and the approximated projected line form an angle α equal to the angle between ideal tangent line and the tangent line from y.
- Note that, so far, the argument was scoped to the tangent line from one of the aggregated points (y in our case). However, we can easily generalize it for approximation on any tangent line, by drawing its parallel line that crosses the point y and apply the same trigonometry on the approximated point (Fig 4 (c)).

Thus, if the approximation happens on a tangent line of point p, where p is between the x, y, the approximation error is smaller than the one obtained using tangent lines at any of the points that are being aggregated. Now, on a half-circle, if we fixate p to be the origin (p_0), assuming the given points' coordinates have a uniform distribution there is a 50% chance that p lies between the aggregated points.

It should be noted as the model gets deeper, the intermediate embeddings of the adjacent nodes/edges tend to show stronger spatial locality properties. Thus, for last layers of a deep neural network, a mix Hyperbolic/Pseudo-Poincaré layer is recommended. we are investigating the optimal strategy in our future work.

Proof of Lemma 2

Given $x \in \mathbb{R}^n$ is a Euclidean equivalent of Poincaré ball's point $p \in \mathbb{B}^n_c$ in the tangent space at origin $\mathcal{T}_{0^n}\mathbb{B}^n_c$. The relation between x and p, as defined by Ganea, Bécigneul, and Hofmann [10] as;

$$p = exp^c_0(x) = \tanh(\sqrt{c}\|x\|)\frac{x}{\sqrt{c}\|x\|},$$

$$x = log_0^c(p) = \tanh^{-1}(\sqrt{c}\|p\|)\frac{p}{\sqrt{c}\|p\|},$$

Simplifying with $\omega(x) = \frac{\tanh(\sqrt{c}\|x\|)}{\sqrt{c}\|x\|}$ and $\omega_{\mathbb{B}_c^n}(p) = \frac{\tanh^{-1}(\sqrt{c}\|p\|)}{\sqrt{c}\|p\|}$ results in:

$$p = \omega(x)x \text{ and } x = \omega_{\mathbb{B}_c^n}(p)p \tag{8}$$

Proof of Lemma 3 and Theorem 1

Let us first look at the case of n = 1, i.e., a single hyperbolic feed-forward layer. This is defined by Ganea, Bécigneul, and Hofmann [10] as;

$$F_1^{\otimes_c}(p) = f_1^{\otimes_c}(p) = exp_0^c(f_1(log_0^c(p))) \tag{9}$$

Reformulating this with Lemma 1,

$$F_1^{\otimes_c}(p) = \omega(f_1(log_0^c(p)))f_1(log_0^c(p));$$

$$\text{Substituting, } x = log_0^c(p);\;\; F_1^{\otimes_c}(p) = \omega(f_1(x))f_1(x); \tag{10}$$

Extending this to $n = 2$,

$$F_2^{\otimes_c}(p) = f_2^{\otimes_c}(f_1^{\otimes_c}(x)) = f_2^{\otimes_c}(\omega(f_1(x))f_1(x))$$

Given that $f_1(x)$ is a linear function;

$$f(ax) = af(x); \tag{11}$$

$$F_2^{\otimes_c}(p) = \omega(f_1(x))f_2^{\otimes_c}(f_1(x));$$

$$F_2^{\otimes_c}(p) = \omega(f_1(x))\omega(f_2(x))f_2(f_1(x));$$

$$F_2^{\otimes_c}(p) = \omega(f_2(x))\omega(f_1(x))F_2(x);$$

Substituting;

$$\Omega(F_2(x)) = \omega(f_1(x))\omega(f_2(x)); \tag{12}$$

$$F_2^{\otimes_c}(p) = \Omega(F_2(x))F_2(x); \tag{13}$$

Extending the above formulation to arbitrary n,

$$F_n^{\otimes_c}(p) = \omega(f_n(x))...\omega(f_2(x))\omega(f_1(x))F_n(x);$$

Reformulating this with Lemma 1, we get the conclusion of Lemma 2;

$$F_n^{\otimes_c}(p) = \exp_0^c(F_n(log_0^c(p))); \tag{14}$$

Substituting;

$$\Omega(F_n(x)) = \omega(f_n(x))...\omega(f_2(x))\omega(f_1(x)); \tag{15}$$

We get the conclusion of Theorem 1;

$$F_n^{\otimes_c}(p) = \Omega(F_n(x))F_n(x);$$

Algorithm

Algorithm 1 provides the generalized algorithm for our proposed hyperbolic model.

Algorithm 1: Normalized hyperbolic model. (Figure 2)

```
Input Euclidean model $F_L$, sample node $x \in \mathbb{R}^n$ Output of Normalized hyperbolic model $F_L^{\otimes_c}(x)$; $f_0^{\otimes_c}(x) = x$;
layer $l : 1 \rightarrow L$  $x \leftarrow f_{l-1}^{\otimes_c}(x)$;
# Layer $l$ of the Euclidean model is $f_l(x)$
$\omega(f_l(x)) = \frac{tanh(\sqrt{c}|f_l(x)|)}{\sqrt{c}|f_l(x)|}$; using Eq. (4) $f_l^{\otimes_c}(x) = \omega(f_l(x))f_l(x)$; using Lemma 3
$F_n^{\otimes_c}(x) = f_L^{\otimes_c}(x)$;
$F_L^{\otimes_c}(x)$
```