

# Modeling Graphs Beyond Hyperbolic: Graph Neural Networks in Symmetric Positive Definite Matrices

Wei Zhao<sup>1,2</sup>(✉), Federico Lopez<sup>1</sup>, \*J. Maxwell Riestenberg<sup>2</sup>, Michael Strube<sup>1</sup>,  
\*Diaaeldin Taha<sup>2</sup>, and Steve Trettel<sup>3</sup>

<sup>1</sup> Heidelberg Institute for Theoretical Studies, Germany  
`{firstname.lastname}@h-its.org`  
<sup>2</sup> Heidelberg University, Germany  
`{mriestenberg, dtaha}@mathi.uni-heidelberg.de`  
<sup>3</sup> University of San Francisco, USA  
`strettel@usfca.edu`

**Abstract.** Recent research has shown that alignment between the structure of graph data and the geometry of an embedding space is crucial for learning high-quality representations of the data. The uniform geometry of Euclidean and hyperbolic spaces allows for representing graphs with uniform geometric and topological features, such as grids and hierarchies, with minimal distortion. However, real-world graph data is characterized by multiple types of geometric and topological features, necessitating more sophisticated geometric embedding spaces. In this work, we utilize the Riemannian symmetric space of symmetric positive definite matrices (SPD) to construct graph neural networks that can robustly handle complex graphs. To do this, we develop an innovative library that leverages the SPD gyrocalculus tools [28] to implement the building blocks of five popular graph neural networks in SPD. Experimental results demonstrate that our graph neural networks in SPD substantially outperform their counterparts in Euclidean and hyperbolic spaces, as well as the Cartesian product thereof, on complex graphs for node and graph classification tasks. We release the library and datasets at <https://github.com/andyweizhao/SPD4GNNs>.

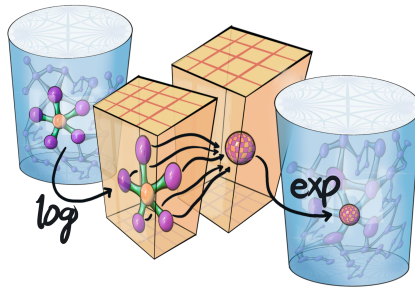
**Keywords:** Graph Neural Networks · Riemannian Geometry · Symmetric Space · Space of Symmetric Positive Definite Matrices

## 1 Introduction

Complex structures are a common feature in real-world graph data, where the graphs often contain a large number of connected subgraphs of varying topologies (including grids, trees, and combinations thereof). While accommodating the diversity of such graphs is necessary for robust representation learning, neither Euclidean nor hyperbolic geometry alone has been sufficient [25].

---

\* These authors contributed equally to this work.



**Fig. 1.** Propagation schema for utilizing SPD geometry while performing calculations in the tangent (Euclidean) space: starting from an SPD embedding, map a node and its neighbors to the tangent space via the logarithm, and perform a modified Euclidean aggregation (Table 1) before returning to SPD via the Riemannian exponential map.

This inefficiency stems from geometric reasons. Properties of the embedding space strongly control which graph topologies embed with low distortion, with simple geometries selecting only narrow classes of graphs. Euclidean geometry provides the foundational example, where its abundant families of equidistant lines allow for efficient representation of grid-like structures, but its polynomial volume growth is too slow to accommodate tree-like data. This is somewhat ameliorated by moving to higher dimensions (with faster polynomial volume growth), though with a serious trade-off in efficiency [4]. An alternative is to move to hyperbolic geometry, where volume growth is exponential, providing plenty of room for branches to spread out for the isometric embedding of trees [7,37]. However, hyperbolic geometry has a complementary trade-off: it does not contain equidistant lines, which makes it unfit for embedding the grid-like structures Euclidean space excelled at [8].

Many proposed graph neural networks utilize representations of graph data to perform machine learning tasks in various graph domains, such as social networks, biology and molecules [24,39,11,13,34,33,14,2]. A subset of these networks take seriously the constraints of geometry on representation capability, and work to match various non-Euclidean geometries to common structures seen in graph data. For instance, Chami et al. [11] and Defferrard et al. [13] show that constructing graph neural networks in hyperbolic and spherical spaces have been successful in embedding graphs with hierarchical and cyclical structures, respectively. However, the relative geometric simplicity of these spaces poses serious limitations including (a) the need to know the graph structure prior to choosing the embedding space, and (b) the inability to perform effectively with graphs built of geometrically distinct sub-structures, a common feature of real-world data.

Avoiding these limitations may necessitate resorting to more complex geometric spaces. For example, Gu et al. [18] employed Cartesian products of various geometric spaces to represent graphs with mixed geometric structures. But any such choice must be carefully considered: isometries play an essential role in the construction of the above architectures, and any increase in complexity accompa-

nied by too great a decrease in symmetry may render a space computationally intractable.

Riemannian symmetric spaces, which have a rich geometry encompassing all the aforementioned spaces, strike an effective balance between geometric generality and ample symmetry. Lopez et al. [27] proposed particular symmetric spaces, namely Siegel spaces, for graph embedding tasks, and demonstrated that many different classes of graphs embed in these spaces with low distortion. Lopez et al. [28] suggested utilizing the symmetric space SPD of symmetric positive definite matrices that is less computationally expensive than Siegel spaces. Furthermore, they developed gyrocalculus tools that enable “vector space operations” on SPD.

Here we extend the idea of Lopez et al. [28] to construct graph neural networks in SPD, particularly by utilizing their gyrocalculus tools to implement the building blocks of graph neural networks in SPD. The building blocks include (a) feature transformation via isometry maps, (b) propagation via graph convolution in the tangent space of SPD (the space of symmetric matrices  $S_n$ ), (c) bias addition via gyrocalculus, (d) non-linearity acting on eigenspace, and (e) three classification layers. We develop *SPD4GNNs*, an innovative library that showcases training five popular graph neural networks in  $SPD_n$ , alongside the functionality for training them in Euclidean and hyperbolic spaces.

We perform experiments to compare four ambient geometries (Euclidean, hyperbolic, products thereof, and SPD) across popular graph neural networks, evaluated on the node and graph classification tasks on nine datasets with varying complexities. Results show that constructing graph neural networks in SPD space leads to big improvements in accuracy over Euclidean and hyperbolic spaces on complex graphs, at the cost of doubling (resp. quadrupling) the training time of graph neural networks compared to hyperbolic space (resp. Euclidean space).

Finally, we provide a summary of the numerical issues we encountered and the solutions to addressing them (see Appendix F).

## 2 Related Work

*Graph Neural Networks.* Graph neural networks (GNNs) have been profiled as the de facto solutions for learning graph embeddings [24,39,40,33,14,2]. These networks can be differentiated into two dimensions: (a) how they propagate information over graph nodes and (b) which geometric space they use to embed the nodes. In **Euclidean space**, a class of GNNs has been proposed that represents graph nodes in a flat space and propagates information via graph convolution in various forms, such as using Chebyshev polynomial filters [12,24], high-order filters [40], importance sampling [20], attention mechanisms [39], graph-isomorphism designs [41,34,2], and differential equations [16,33,14]. In contrast, **non-Euclidean spaces** have a richer structure for representing geometric graph structures in curved spaces. Recently, there has been a line of GNNs developed in these spaces that performs graph convolution on different Riemannian manifolds in order to accommodate various graph structures, such as hyperbolic space on

tree-like graphs [11,26,42], spherical space on spherical graphs [13], and Cartesian products of thereof [19].

*SPD Space.* Representing data with SPD matrices has been researched for many years, with the representations being primarily in the form of covariance matrices [15,22,44,17,9]. These matrices capture the statistical dependencies between Euclidean features. Recent research focused on designing the building blocks of neural networks in the space of covariance matrices. This includes feature transformation that maps Euclidean features to covariance matrices via geodesic Gaussian kernels [15,6], nonlinearity on the eigenvalues of covariance matrices [22], convolution through SPD filters [44] and Frechét mean [9], Riemannian recurrent networks [10], and Riemannian batch normalization [5].

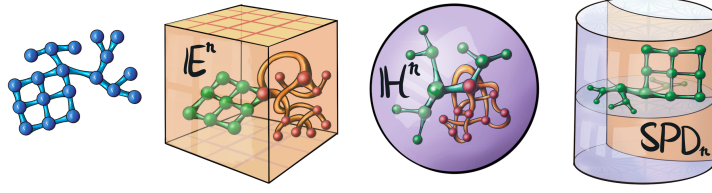
Nguyen et al. [31] recently approached hand gesture classification by embedding graphs into SPD via a neural network. The architectures we consider here are different. While we alternate between SPD and its tangent space using the exponential and logarithm maps, Nguyen et al. [31] do so via an aggregation operation and the log map. Further, we couple this alternation with our building blocks to operate graph neural networks in SPD.

### 3 Background

#### 3.1 The Space SPD

We let  $\text{SPD}_n$  denote the space of positive definite real symmetric  $n \times n$  matrices. This space has the structure of a Riemannian manifold of non-positive curvature of  $n(n+1)/2$  dimensions. The tangent space to any point of  $\text{SPD}_n$  can be identified with the vector space  $S_n$  of all real symmetric  $n \times n$  matrices.  $\text{SPD}_n$  is more flexible than Euclidean or hyperbolic geometries, or products thereof. In particular, it contains  $n$ -dimensional Euclidean subspaces,  $(n-1)$ -dimensional hyperbolic subspaces, products of  $\lfloor \frac{n}{2} \rfloor$  hyperbolic planes, and many other interesting spaces as totally geodesic submanifolds; see the reference [21] for an in-depth introduction to these well-known facts. While it is not yet fully understood how our proposed models leverage the Euclidean and hyperbolic subspaces in  $\text{SPD}_n$ , we hypothesize that the presence of these subspaces is an important factor in the superior performance of  $\text{SPD}_n$  graph neural networks. Refer to Figure 2 for a demonstration of how this hypothesis may manifest.

*Riemannian Exponential and Logarithmic Maps.* For  $\text{SPD}_n$ , the Riemannian exponential map at the basepoint  $I_n$  agrees with the standard matrix exponential  $\exp: S_n \rightarrow \text{SPD}_n$ . This map is a diffeomorphism with inverse the matrix logarithm  $\log: \text{SPD}_n \rightarrow S_n$ . These maps allow us to pass from  $\text{SPD}_n$  to  $S_n$  and back again. Given any two points  $X, Y \in \text{SPD}_n$ , there exists an isometry (i.e., a distance-preserving transformation) that maps  $X$  to  $Y$ . As such, the choice of a basepoint for the exponential and logarithm maps is arbitrary since any other point can be mapped to the basepoint by an isometry. In particular, there is no loss of generality with fixing the basepoint  $I_n$  as we do.



**Fig. 2.** Graphs exhibiting both euclidean/grid-like and hyperbolic/tree-like features (left) cannot embed well in either euclidean or hyperbolic spaces due to the impossibility of isometrically embedding trees in Euclidean spaces and grids in hyperbolic spaces (center). However,  $\text{SPD}_n$  (right) contains both euclidean and hyperbolic subspaces, which allows embedding a broad class of graphs, including the example in the figure.

*Non-linear Activation Functions in  $\text{SPD}_n$ .* We use two non-linear functions on SPD matrices, namely (i) ReEig [22]: factorizing a point  $P \in \text{SPD}_n$  and then employing the ReLU-like non-linear activation function  $\varphi_a$  to suppress the positive eigenvalues that are bigger than 0.5<sup>2</sup>:

$$\varphi^{\text{SPD}}(P) = U \varphi_a(\Sigma) U^T \quad P = U \Sigma U^T \quad (1)$$

(ii) TgReEig: projecting  $P \in \text{SPD}_n$  into the tangent space and then suppressing the negative eigenvalues of the projected point  $\in S_n$  with the ReLU non-linear activation function  $\varphi_b$ , i.e.  $\varphi^{\text{SPD}}(P) = U \exp(\varphi_b(\log(\Sigma))) U^T$ .

### 3.2 Gyrocalculus on SPD

*Addition and Subtraction.* Gyro-calculus is a way of expressing natural analogues of vector space operations in Riemannian manifolds. Following Lopez et al. [28], given two points  $P, Q \in \text{SPD}_n$ , we denote gyro-addition and gyro-inversion by:

$$P \oplus Q = \sqrt{P} Q \sqrt{P} \quad \ominus P = P^{-1} \quad (2)$$

For  $P, Q \in \text{SPD}_n$ , the value  $P \oplus Q \in \text{SPD}_n$  is the result of applying the  $\text{SPD}_n$ -translation moving the basepoint  $I_n$  to  $P$ , evaluated on  $Q$ .

*Isometry Maps.* Any invertible  $n \times n$  real matrix  $M \in \text{GL}(n, \mathbb{R})$  defines an isometry of  $\text{SPD}_n$  by

$$M \odot P = M P M^T \quad (3)$$

where  $P \in \text{SPD}_n$ .

Lopez et al. [28] proposed defining  $M$  in two forms, namely a rotation element in  $SO(n)$  and a reflection element in  $O(n)$ . In this case, the choice of rotation and reflection becomes a hyperparameter for  $M$ , and that needs to be selected before training. In contrast, the form of  $M$  we considered is more flexible and can be automatically adjusted by training data. To do so, we first let  $M$  denote the

<sup>2</sup> TgReEig equals ReEig in the case of  $\varphi_a(x) = \max(x, 1)$ .

Operations	GNNs	Euclidean Space	Hyperbolic and SPD Space
Feature Trans	All	$ h_i^l = W^l x_i^{l-1}$	$ Q_i^l = M^l \odot Z_i^{l-1}$
Propagation	GCN	$ p_i^l = \sum_{j \in \mathcal{N}(i)} k_{i,j} h_j^l$	$ P_i^l = \exp(\sum_{j \in \mathcal{N}(i)} k_{i,j} \log(Q_j^l))$
	GAT	$ p_i^l = \alpha_{i,i} h_i^l + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} h_j^l$	$ P_i^l = \exp(\alpha_{i,i} \log(Q_i) + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} \log(Q_j^l))$
	Cheb	$ p_i^l = h_i^l + W^l \sum_{j \in \mathcal{N}(i)} k_{i,j} h_j^l$	$ P_i^l = Q_i^l \oplus (M^l \odot \exp(\sum_{j \in \mathcal{N}(i)} k_{i,j} \log Z_j^{l-1}))$
Bias&Nonlin	All	$ x_i^l = \varphi(p_i^l + b^l)$	$ Z_i^l = \varphi(P_i^l \oplus B^l)$

**Table 1.** Comparison of operations in different spaces across three graph neural networks, i.e., GCN [24], GAT [39] and 1-order Cheb [12]. SGC [40] and GIN [41] are presented in Appendix B, which indeed apply propagation before feature transformation.

orthogonal basis of a learnable square matrix, and then tune the square matrix from training data. Thus,  $M$ , as an orthogonal matrix that extends rotations and reflections, is better suited to fit the complexity of graph data.

## 4 Graph Neural Networks

In this section, we introduce the notation and building blocks of graph neural networks using graph convolutional network (GCN) [24] as an example, and present modifications for operating these building blocks in SPD. Table 1 establishes parallels between five popular graph neural networks in Euclidean, hyperbolic and SPD spaces.

### 4.1 GCN in Euclidean Space

Given a graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  with a vertex set  $\mathcal{V}$  and an edge set  $\mathcal{E}$ , we define  $d$ -dimensional input node features  $(x_i^0)_{i \in \mathcal{V}}$ , where the superscript 0 indicates the first layer. The goal of a graph neural network is to learn a mapping denoted by:

$$f : (\mathcal{V}, \mathcal{E}, (x_i^0)_{i \in \mathcal{V}}) \rightarrow \mathcal{Z} \in \mathbb{R}^{|\mathcal{V}| \times d}$$

where  $\mathcal{Z}$  is the space of node embeddings obtained from the final layer of GCN, which we take as the input of classification layer to perform downstream tasks.

Let  $\mathcal{N}(i) = \{j : (i, j) \in \mathcal{E}\} \cup \{i\}$  be the set of neighbors of  $i \in \mathcal{V}$  with self-loops, and  $(W^l, b^l)$  be a matrix of weights and a vector of bias parameters at layer  $l$ , and  $\varphi(\cdot)$  be a non-linear activation function. We now introduce **message passing**, which consists of the following three components for exchanging information between the node  $i$  and its neighbors at layer  $l$ :

$$h_i^l = W^l x_i^{l-1} \quad \text{Feature transform} \quad (4)$$

$$p_i^l = \sum_{j \in \mathcal{N}(i)} k_{i,j} h_j^l \quad \text{Propagation} \quad (5)$$

$$x_i^l = \varphi(p_i^l + b^l) \quad \text{Bias \& Nonlinearity} \quad (6)$$

where  $k_{i,j} = c_i^{-\frac{1}{2}} c_j^{-\frac{1}{2}}$  with  $c_i$  as the cardinality of  $\mathcal{N}(i)$ .  $k_{i,j}$  represents the relative importance of the node  $j$  to the node  $i$ .

## 4.2 GCN in SPD

*Mapping from Euclidean to SPD space.* Oftentimes, input node features are not given in SPD, but in Euclidean space  $(x_i^0)_{i \in \mathcal{V}} \in \mathbb{R}^d$ . Therefore, we design a transformation that maps Euclidean features to a point in SPD. To do so, we first learn a linear map that transforms the  $d$ -dimensional input features into a vector of dimension  $n(n+1)/2$ , that we arrange as the upper triangle of an initially zero matrix  $A \in \mathbb{R}^{n \times n}$ . We then define a symmetric matrix  $U \in S_n$  such that  $U = A + A^T$ . We now apply the exponential map such that  $Z = \exp(U)$ , which moves the coordinates from the tangent space  $S_n$  to the original manifold  $\text{SPD}_n$ . Thus, the resulting node embeddings  $(Z_i^0)_{i \in \mathcal{V}}$  are in  $\text{SPD}_n$ . By performing this mapping only once, we enable GNNs to operate in  $\text{SPD}_n$ .

*Feature Transform.* We apply isometry maps to transform points in SPD at different layers, denoted by:  $Q_i^l = M^l \odot Z_i^{l-1}$ , where  $Q_i^l, Z_i^{l-1} \in \text{SPD}_n$  and  $M^l$  is a isometry map (see §3.2) at layer  $l$  of the GNN.

*Propagation.* This step aggregates information from all the neighbors  $\mathcal{N}(i)$  of a given node  $i$ , with the information weighted by the importance of a neighbor to the node  $i$  (see Eq. 5). We note that propagation involves addition and scaling operators. This results in two alternative approaches for computing propagation: (a) employing gyro-addition to aggregate information over the neighbors for each node; (b) computing the Riemannian Fréchet mean in  $\text{SPD}_n$ —which requires hundreds of iterations to find a geometric center. Therefore, these approaches are costly to compute and also involve the use of cumbersome Riemannian optimization algorithms (see Appendix A for optimization). Here we perform aggregation via graph convolution in the space of symmetric matrices  $S_n$  denoted by:  $P_i^l = \exp(\sum_{j \in \mathcal{N}(i)} k_{i,j} \log(Q_j^l))$ , where  $P_i^l \in \text{SPD}_n$  and  $k_{i,j} = c_i^{-\frac{1}{2}} c_j^{-\frac{1}{2}}$  (as in the Euclidean case). This is similar to the approach of Chami et al. [11] by performing propagation in the tangent pace and the posterior projection through the exponential map.

*Bias Addition and Non-linearity.* Finally, we add the bias  $B^l$  at layer  $l$  to the result of propagation through gyro-addition followed by applying a non-linear function, denoted by:  $Z_i^l = \varphi^{\text{SPD}}(P_i^l \oplus B^l)$ , with  $Z_i^l \in \text{SPD}_n$  as the new embedding for the node  $i$  at layer  $l$  and  $B^l \in \text{SPD}_n$ .

*Message Passing in SPD.* We establish a one-to-one correspondence between the Euclidean and SPD versions of GCN at layer  $l$  for node  $i$ :

$$Q_i^l = M^l \odot Z_i^{l-1} \quad \text{Feature transform} \quad (7)$$

$$P_i^l = \exp\left(\sum_{j \in \mathcal{N}(i)} k_{i,j} \log(Q_j^l)\right) \quad \text{Propagation} \quad (8)$$

$$Z_i^l = \varphi^{\text{SPD}}(P_i^l \oplus B^l) \quad \text{Bias \& non-lin} \quad (9)$$

*Classification.* In node classification setups<sup>3</sup>, we are given  $\{Z_i, y_i\}_{i=1}^N$  on a dataset, with  $N$  as the number of instances,  $Z_i \in \text{SPD}_n$  as the  $i$ -th node embedding obtained from the final layer of a graph neural network, and  $y_i \in \{1, \dots, K\}$  as the true class of  $i$ -th node. Let  $h : \text{SPD} \mapsto \{1, \dots, K\}$  be a classifier that best predicts the label  $y_i$  of a given input  $Z_i$ . Indeed, the input space of  $h$  can be in various forms, not limited to  $\text{SPD}_n$ . Here we introduce three classifiers in two alternative input spaces: (a)  $\mathbb{R}^{d(d+1)/2}$  and (b)  $S_n$ <sup>4</sup>. To do so, we first take Riemannian logarithm  $\log : \text{SPD}_n \rightarrow S_n$  of each  $Z_i$  at the identity. For (a), we vectorize the upper triangle elements of  $X$  as  $\mathbf{x} = (X_{1,1} \cdots X_{1,d}, X_{2,2}, \dots, X_{d,d}) \in \mathbb{R}^{d(d+1)/2}$ , and then design two classifiers, i.e., LINEAR-XE (Linear Classifier coupled with Cross-Entropy loss) and NC-MM (Nearest Centroid Classifier with Multi-Margin loss). For (b), we design a SVM-like classifier SVM-MM acting in  $S_n$ , a similar approach to the proposal of Nguyen et al. [31]. We present the details of these classifiers in Appendix G.

## 5 Experiments

In this section, we first perform experiments for node and graph classification, and then analyze the ability of three geometric spaces in arranging and separating nodes with different classes. Further, we compare the training efficiency of different spaces and the usefulness of three classifiers. Lastly, we discuss **product space** (the Cartesian product of Euclidean and hyperbolic spaces) and compare it with SPD in Appendix H.

*Baselines.* To investigate the usefulness of different geometric spaces on graph neural networks, we choose five well-known graph architectures as representatives: GCN [24], GAT [39], Cheb [12], SGC [40] and GIN [41], and evaluate these architectures in Euclidean, hyperbolic and SPD spaces. For the hyperbolic versions of these architectures, we use Poincaré models and extend the implementation of Poincaré GCN [11] to other four architectures.

### 5.1 Node Classification

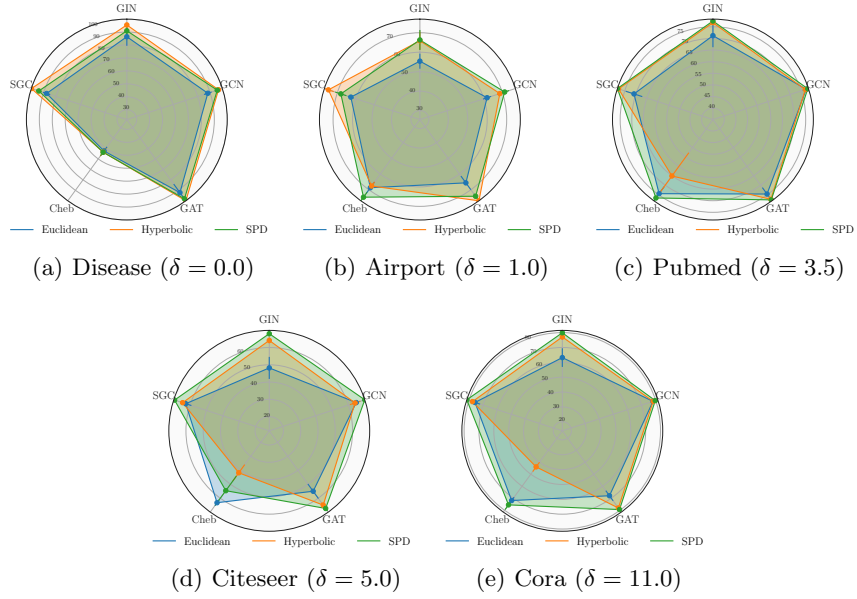
*Datasets.* We evaluate graph neural networks in the three spaces on 5 popular datasets for node classification: Disease [1], Airport [43], Pubmed [30], Citeseer and Cora [36]. Overall, each dataset has a single graph that contains up to thousands of labeled nodes. We use the public train, validation and test splits of each dataset, and provide dataset statistics in Appendix C. Unlike previous works [11,42], we only use original node features to ensure a fair and transparent comparison.

<sup>3</sup> For graph classification,  $Z_i$  and  $y_i$  denote the ‘center’ of the graph  $i$  and its true class. We take the arithmetic mean of node embeddings in  $\text{SPD}_n$  to produce  $Z_i \in \text{SPD}_n$ .

<sup>4</sup> We also design several classifiers with the input space in  $\text{SPD}_n$ , but these do not yield better results than those in  $S_n$ .

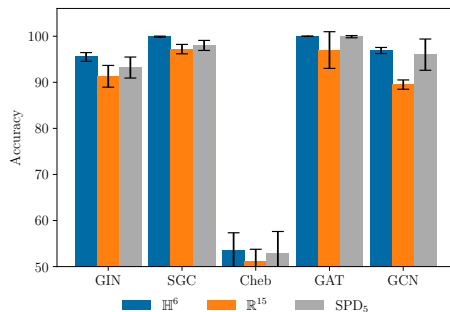


*Setup.* We compare three geometries, namely Euclidean, hyperbolic and SPD, in two low-dimensional spaces: (i) 6 dimensions:  $\mathbb{R}^6$ ,  $\mathbb{H}^6$  and  $\text{SPD}_3$ , and (ii) 15 dimensions:  $\mathbb{R}^{15}$ ,  $\mathbb{H}^{15}$  and  $\text{SPD}_5$ , a common choice of dimensions in previous work [11,42]. The reason we considered for low-dimensional space is the following: If the structure of data matches the geometry of embedding space, a low-dimensional space can be leveraged efficiently for producing high-quality embeddings. If they do not match, a large dimension is needed to compensate for the wrong use of unsuitable geometric spaces. Here we investigate the efficiencies of different geometries in space use when given a small dimension. We report mean accuracy and standard deviation of binary/multi-label classification results under 10 runs, and provide training details in Appendix A.



**Fig. 3.** Evaluation of five graph neural networks coupled with LINEAR-XE on five node classification datasets in the three 6-dimensional spaces:  $\mathbb{R}^6$ ,  $\mathbb{H}^6$  and  $\text{SPD}_3$ . Each radar chart shows classification accuracy (on a varying scale, as noted by the gridlines with circular shapes) from the five GNN architectures on a dataset. Each dataset has only one graph.  $\delta$ -hyperbolicity shows the degree to which the dataset graph is a hyperbolic tree. A smaller  $\delta$  indicates a more tree-like dataset.

*Results.* Figure 3 shows the accuracy results of a node classification task in the three 6-dimensional geometries across five datasets on five GNNs, see also Table 6. For graphs with  $\delta$ -hyperbolicity  $> 1$ ,  $\text{SPD}_3$  achieves the best accuracy in all cases except the Cheb architecture on the Citeseer dataset. We also observe that the accuracy of SPD is similar to hyperbolic space on the Airport dataset  $\delta = 1$ .



**Fig. 4.** Comparison of 6d and 15d spaces on Disease.  $\text{SPD}_5$  has 15 dimensions.

The Disease graph is a tree ( $\delta = 0$ ) and has optimal performance in hyperbolic space. The accuracy is much lower for these two tree-like datasets in Euclidean geometry for all GNNs except Cheb.

In the case of tree-like datasets, hyperbolic space provides not only accuracy for these tasks but also efficiency. Figure 4 compares 6-dimensional hyperbolic space to  $\mathbb{R}^{15}$  and  $\text{SPD}_5$  (also 15-dimensional), showing that even a much smaller dimensional hyperbolic space achieves the best performance on Disease. Notably, the poor performance of Cheb across all spaces might be attributed to the low representational capacity of the first-order Chebyshev polynomial used in the graph neural network for embedding the tree structure of Disease. Results comparing the 6d and 15d geometries are reported in Table 7 (appendix).

## 5.2 Graph Classification

*Datasets.* We evaluate graph neural networks in three spaces on the popular TUDataset benchmark [29]. Here we focus on datasets with node features, and choose a sample of 4 popular datasets in two domains, namely (a) Biology: ENZYMES [35] and PROTEINS [3]; (b) Molecules: COX2 [38] and AIDS [32]. Overall, each dataset instance has one labeled graph with dozens of nodes. We use the first split of train and test sets in the 10-fold cross-validation setup<sup>5</sup>, and select 10% of the training set uniformly at random as the development set. We provide data statistics in Appendix C.

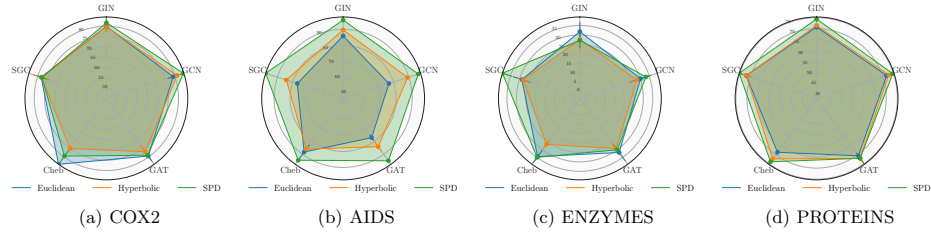
*Setup.* Following Morris et al. [29], we predict the class of an unlabeled graph by classifying its center. In particular, we produce the graph center by using mean pooling to take the arithmetic mean of node embeddings in a graph. To compare efficiency in space use, we conduct experiments in three spaces with the same dimension size of 36, namely  $\mathbb{R}^{36}$ ,  $\mathbb{H}^{36}$  and  $\text{SPD}_8$ , the smallest dimension

<sup>5</sup> Morris et al. [29] proposed to run 10-fold cross-validation by generating ten splits of train and test sets, and repeat this ten times to reduce model initialization. This requires 100 runs for one setup, but it is impractical in our large-scale study.

size in the grid search from Morris et al. [29]. We report the mean accuracy and standard deviation of graph classification results under 10 runs, and provide training details in Appendix A.

*Results.* Figure 5 shows the accuracy results of a graph classification task in three geometries across five datasets on five GNNs, see also Table 8. Figure 6 shows the distribution of  $\delta$ -hyperbolicity over instances. We see that  $\text{SPD}_8$  achieves better or similar accuracy than its counterparts of the same dimension in all cases except Cheb on COX2 and GIN on ENZYMES. On the AIDS dataset, SPD achieves much better accuracy across all GNNs, and on ENZYMES SPD achieves much better accuracy on SGC.

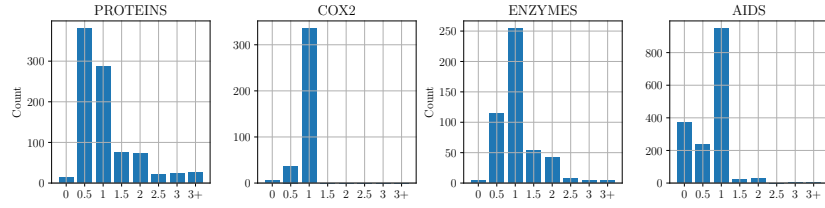
We also observe that hyperbolic space does not yield much increased accuracy over Euclidean space in most cases, except the AIDS data set. Furthermore, SPD significantly outperforms hyperbolic space on the AIDS dataset but not the COX2 data set. Both of these datasets have the property that almost all instances are tree-like ( $\delta \leq 1$ ) (see Figure 6), but the hyperbolicity constants are less concentrated in the AIDS dataset than in the COX2 dataset. It is possible that the flexibility of SPD explains the increased performance over hyperbolic space in this case. For example, SPD admits totally geodesic submanifolds isometric to hyperbolic spaces of varying constant curvatures. It would be interesting to find out, for example, if these graphs of different hyperbolicity constants stay near copies of hyperbolic space of different curvatures.



**Fig. 5.** Evaluation of five graph neural networks with LINEAR-XE on four graph classification datasets in the three 36-dimensional spaces:  $\mathbb{R}^{36}$ ,  $\mathbb{H}^{36}$  and  $\text{SPD}_8$ .

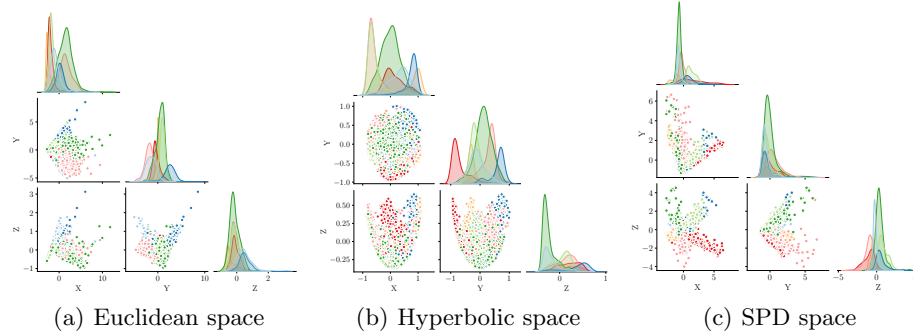
### 5.3 Analysis

*Class Separation.* Figure 7 shows a visualization of node embeddings obtained from the final layer of SGConv on Cora. In each space, we vectorize node embeddings, and then use PCA to extract the top 3 dimensions. We then look at the projections to that 3-dimensional space by further projecting to the x-y, y-z, and x-z planes. For instance, the x-y plane is the projection to the top 2 dimensions of PCA. The x-x, y-y and z-z planes show the informativeness of each dimension in terms of class separation.



**Fig. 6.** Distributions of  $\delta$ -hyperbolicity on four graph classification datasets, where each instance consists of one graph. Y-axis shows the number of graphs for a given  $\delta$ -hyperbolicity on X-axis.

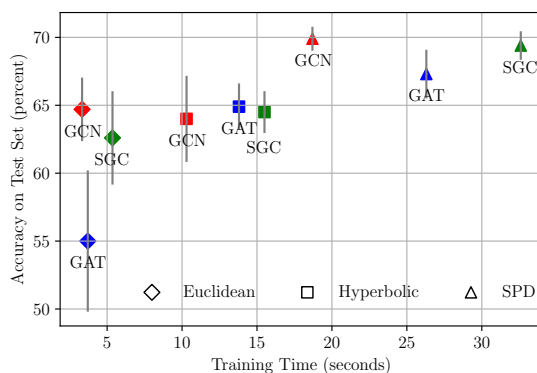
Figure 7 (a) depicts the case when the ambient geometry is Euclidean. In this example, nodes from the pink, blue and green classes are well-separated but the nodes in red and orange cannot be easily distinguished. Figure 7 (b) depicts the case when the nodes are embedded in the Poincaré ball model of hyperbolic space. In the x-z plane five classes are well-separated, including the red and orange classes. Figure 7 (c) depicts the case when the nodes are embedded into SPD where the best class separation is achieved. Indeed, the Cora graph has hyperbolicity constant  $\delta = 11$ , so one cannot expect it to embed well into hyperbolic space.



**Fig. 7.** Visualizations of the node embeddings into three 6-dimensional geometries for SGC on the Cora dataset. In each space, the nodes are vectorized and then projected linearly to  $\mathbb{R}^3$  via PCA.

*Training Time.* As a case study on the impact of the choice of the latent space geometry on the training time of GNN models, Figure 8 compares the training efficiency of three graph neural networks on Citeseer across three 6-dimensional manifolds:  $\mathbb{R}^6$ ,  $\mathbb{H}^6$ , and  $\text{SPD}_3$ . Overall, we see that models with Euclidean latent spaces needed the least amount of training time, but produced the lowest accuracy.

Moreover, hyperbolic space slows down the training in Euclidean space by up to four times, and the effect on accuracy is either slightly positive or negative. For instance, using hyperbolic space with the SGC architecture only brought small accuracy improvements, while applying it to GCN resulted in a drop in accuracy. On the other hand, even though SPD models require nearly double the training time of hyperbolic models, the SPD models bring big improvements in accuracy, not only on Citeseer in the current study case, but also on many datasets such as AIDS and PROTEINS (see Figure 5). We note that the longer training times for SPD models can be attributed to the involvement of eigendecompositions. However, the benefits of using SPD space in graph neural networks appear to outweigh this drawback.



**Fig. 8.** Evaluation of three 6-dimensional spaces across graph neural networks in terms of training time and accuracy on Citeseer ( $\delta = 5.0$ ). Each point has a unique pattern that combines color and shape. For instance, a red triangular means GCN in SPD.

*Classifiers.* Our three classification layers are built upon traditional classification methods. LINEAR-XE and SVM-MM are both linear classifiers that separate classes with hyperplanes, differing in the choice of loss functions: cross-entropy and multi-margin loss. In contrast, NC-MM layer learns class-specific centroids and then determining the class of an unlabeled node (or graph) by examining which centroid it is closest to according to a similarity function.

Table 2 shows the usefulness of our classifiers in SPD on Citeseer and Cora. Overall, we see that the MM-based classifiers (SVM-MM and NC-MM) are often helpful, outperforming LINEAR-XE in SPD, and when they succeed, their improvements are substantial. This means the benefits of using SPD and these advanced classifiers are complementary, resulting in stacked performance gains. This is an important finding as it hints at the possibility of accommodating more advanced classification methods recently developed in Euclidean space, when constructing graph neural networks in SPD. Note that the results on other datasets are similar, which we present in Appendix D and E.

**Table 2.** Comparison of different classifiers on Citeseer (top) and Cora (bottom). We bold the best accuracy in each row.

	$\mathbb{R}^6$	$\text{SPD}_3$		
	LIN-XE	LIN-XE	SVM-MM	NC-MM
GIN	$48.2 \pm 6.3$	<b><math>68.0 \pm 1.3</math></b>	$67.3 \pm 1.2$	$67.0 \pm 0.8$
SGC	$62.6 \pm 3.4$	$69.4 \pm 1.0$	<b><math>69.7 \pm 0.8</math></b>	$67.9 \pm 1.5$
Cheb	$63.2 \pm 2.1$	$54.6 \pm 10.4$	$61.4 \pm 4.4$	<b><math>64.0 \pm 2.3</math></b>
GAT	$55.0 \pm 5.2$	$67.3 \pm 1.7$	<b><math>69.2 \pm 0.7</math></b>	$68.1 \pm 1.1$
GCN	$64.7 \pm 2.3$	<b><math>69.9 \pm 0.8</math></b>	$69.2 \pm 0.8$	$68.2 \pm 1.0$
GIN	$77.1 \pm 1.0$	<b><math>79.9 \pm 0.6</math></b>	$79.5 \pm 0.6$	$78.8 \pm 1.0$
SGC	$75.7 \pm 3.6$	$81.5 \pm 0.9$	<b><math>81.8 \pm 0.3</math></b>	$81.1 \pm 0.6$
Cheb	$71.9 \pm 2.8$	$75.5 \pm 3.9$	$77.9 \pm 2.4$	<b><math>79.2 \pm 1.3</math></b>
GAT	$67.9 \pm 4.2$	$79.4 \pm 0.9$	<b><math>81.2 \pm 1.4</math></b>	<b><math>81.2 \pm 1.1</math></b>
GCN	$78.1 \pm 1.7$	$79.7 \pm 0.9$	<b><math>80.7 \pm 0.5</math></b>	$80.2 \pm 1.3$

## 6 Conclusions

This work brings sophisticated geometric tools to graph neural networks (GNNs). Following the maxim ‘complex data requires complex geometry’, we leverage the flexibility of the space of symmetric positive definite (SPD) matrices to construct GNNs which do not require careful prior knowledge of graph topologies. This is a distinct advantage over familiar spaces such as Euclidean, spherical or hyperbolic geometries, where only narrow classes of graphs embed with low distortion.

To operate GNNs in SPD, we designed several building blocks, and developed a library (SPD4GNN) that enables training five popular GNNs in SPD, Euclidean and hyperbolic spaces. Our results confirm the strong connection between graph topology and embedding geometry: GNNs in SPD provide big improvements on graph datasets with multi-modal structures, with their counterparts in hyperbolic space performing better on strictly tree-like graphs.

Determining the optimal classifier for training GNNs in the complex geometry of SPD is challenging, and presents an avenue for continued improvement. This work only begins the process of designing geometrically meaningful classifiers and identifying the conditions which guarantee good performance. Additional performance gains may come through careful implementation of the computationally demanding functions in SPD. While this work contains techniques for accelerating computations in SPD, further optimization is likely possible.

Constructing tools to aid the interpretability of SPD embeddings is an important direction of future work, including quantitative measures for (a) comparing the geometry of the learned embeddings to the real-world graphs’ topology and (b) understanding how the geometric features of SPD are leveraged in for graph tasks. While the results of this current work suggest some of SPD’s superior performance may be due to graphs of varying hyperbolicity finding geometric subspaces optimally adapted to their curvature, such measures would enable the precise quantitative analysis required for verification.

## 7 Ethical Considerations

We have not identified any immediate ethical concerns such as bias and discrimination, misinformation dissemination, privacy issues, originating from the contributions presented in this work. However, it is important to note that our SPD models use computationally demanding functions, such as determining eigenvalues and eigenvectors, which may incur a negative environmental impact due to increased energy consumption. Nevertheless, SPD models do not outperform Euclidean and hyperbolic counterparts in terms of computational overhead. This is because Euclidean and hyperbolic models would require substantial computing resources when dealing with larger dimensions, a necessity for compensating for the challenges of embedding complex graphs into these ill-suited spaces.

## Acknowledgements

We thank Anna Wienhard and Maria Beatrice Pozetti for insightful discussions, as well as the anonymous reviewers for their thoughtful feedback that greatly improved the texts. This work has been supported by the Klaus Tschira Foundation, Heidelberg, Germany, as well as under Germany’s Excellence Strategy EXC-2181/1 - 390900948 (the Heidelberg STRUCTURES Cluster of Excellence).

## References

1. Anderson, R.M., May, R.M.: Infectious diseases of humans: dynamics and control. Oxford university press (1992)
2. Barcelo, P., Galkin, M., Morris, C., Orth, M.R.: Weisfeiler and leman go relational. In: The First Learning on Graphs Conference (2022), [https://openreview.net/forum?id=wY\\_IYhh6pqj](https://openreview.net/forum?id=wY_IYhh6pqj)
3. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. *Bioinformatics* **21**(suppl.1), i47–i56 (2005)
4. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* **34**(4), 18–42 (2017)
5. Brooks, D., Schwander, O., Barbaresco, F., Schneider, J.Y., Cord, M.: Riemannian batch normalization for SPD neural networks. In: Wallach, H., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*. vol. 32, pp. 15489–15500. Curran Associates, Inc. (2019), <https://proceedings.neurips.cc/paper/2019/file/6e69ebbfad976d4637bb4b39de261bf7-Paper.pdf>
6. Brooks, D.A., Schwander, O., Barbaresco, F., Schneider, J.Y., Cord, M.: Exploring complex time-series representations for riemannian machine learning of radar data. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. pp. 3672–3676 (2019). <https://doi.org/10.1109/ICASSP.2019.8683056>
7. Buyalo, S., Schroeder, V.: Embedding of hyperbolic spaces in the product of trees. *Geometriae Dedicata* **113**(1), 75–93 (2005)

8. Cannon, J.W., Floyd, W.J., Kenyon, R., Parry, W.R., et al.: Hyperbolic geometry. *Flavors of geometry* **31**(59-115), 2 (1997)
9. Chakraborty, R., Bouza, J., Manton, J., Vemuri, B.C.: Manifoldnet: A deep neural network for manifold-valued data with applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence* pp. 1–1 (2020). <https://doi.org/10.1109/TPAMI.2020.3003846>
10. Chakraborty, R., Yang, C.H., Zhen, X., Banerjee, M., Archer, D., Vaillancourt, D., Singh, V., Vemuri, B.: A statistical recurrent model on the manifold of symmetric positive definite matrices. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*. vol. 31. Curran Associates, Inc. (2018), <https://proceedings.neurips.cc/paper/2018/file/7070f9088e456682f0f84f815ebda761-Paper.pdf>
11. Chami, I., Ying, Z., Ré, C., Leskovec, J.: Hyperbolic graph convolutional neural networks. In: *Advances in Neural Information Processing Systems* 32, pp. 4869–4880. Curran Associates, Inc. (2019), <https://proceedings.neurips.cc/paper/2019/file/0415740eaa4d9dec8da001d3fd805f-Paper.pdf>
12. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems* **29** (2016)
13. Defferrard, M., Milani, M., Gusset, F., Perraudin, N.: DeepSphere: A graph-based spherical CNN. In: *International Conference on Learning Representations* (2020), <https://openreview.net/forum?id=B1e301StPB>
14. Di Giovanni, F., Rowbottom, J., Chamberlain, B.P., Markovich, T., Bronstein, M.M.: Graph neural networks as gradient flows. *arXiv preprint arXiv:2206.10991* (2022)
15. Dong, Z., Jia, S., Zhang, C., Pei, M., Wu, Y.: Deep manifold learning of symmetric positive definite matrices with application to face recognition. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. p. 4009–4015. AAAI’17, AAAI Press (2017)
16. Eliasof, M., Haber, E., Treister, E.: Pde-gcn: Novel architectures for graph neural networks motivated by partial differential equations. *Advances in Neural Information Processing Systems* **34**, 3836–3849 (2021)
17. Gao, Z., Wu, Y., Bu, X., Yu, T., Yuan, J., Jia, Y.: Learning a robust representation via a deep network on symmetric positive definite manifolds. *Pattern Recognition* **92**, 1–12 (2019). <https://doi.org/https://doi.org/10.1016/j.patcog.2019.03.007>, <https://www.sciencedirect.com/science/article/pii/S0031320319301062>
18. Gu, A., Sala, F., Gunel, B., Ré, C.: Learning mixed-curvature representations in product spaces. In: *International Conference on Learning Representations* (2019), <https://openreview.net/forum?id=HJxeWnCcF7>
19. Gu, A., Sala, F., Gunel, B., Ré, C.: Learning mixed-curvature representations in product spaces. In: *International Conference on Learning Representations* (2019), <https://openreview.net/forum?id=HJxeWnCcF7>
20. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*. vol. 30. Curran Associates, Inc. (2017), <https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf>
21. Helgason, S.: *Differential geometry, Lie groups, and symmetric spaces*. Academic Press New York (1978)



22. Huang, Z., Van Gool, L.: A Riemannian network for SPD matrix learning. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. p. 2036–2042. AAAI’17, AAAI Press (2017)
23. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations **abs/1412.6980** (2014)
24. Kipf, T.N., Welling, M.: 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 (2017), <https://openreview.net/forum?id=SJU4ayYgl>
25. Krioukov, D., Papadopoulos, F., Kitsak, M., Vahdat, A., Boguná, M.: Hyperbolic geometry of complex networks. *Physical Review E* **82**(3), 036106 (2010)
26. Liu, Q., Nickel, M., Kiela, D.: Hyperbolic graph neural networks. In: Wallach, H., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*. vol. 32. Curran Associates, Inc. (2019)
27. López, F., Pozzetti, B., Trettel, S., Strube, M., Wienhard, A.: Symmetric spaces for graph embeddings: A finsler-riemannian approach. In: Meila, M., Zhang, T. (eds.) *Proceedings of the 38th International Conference on Machine Learning. Proceedings of Machine Learning Research*, vol. 139, pp. 7090–7101. PMLR (18–24 Jul 2021), <http://proceedings.mlr.press/v139/lopez21a.html>
28. López, F., Pozzetti, B., Trettel, S., Strube, M., Wienhard, A.: Vector-valued distance and gyrocalculus on the space of symmetric positive definite matrices. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) *Advances in Neural Information Processing Systems*. vol. 34. Curran Associates, Inc. (2021)
29. Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M.: Tudataset: A collection of benchmark datasets for learning with graphs. In: *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)* (2020), [www.graphlearning.io](http://www.graphlearning.io)
30. Namata, G., London, B., Getoor, L., Huang, B., Edu, U.: Query-driven active surveying for collective classification. In: 10th International Workshop on Mining and Learning with Graphs. vol. 8, p. 1 (2012)
31. Nguyen, X.S., Brun, L., Lezoray, O., Bougleux, S.: A neural network based on SPD manifold learning for skeleton-based hand gesture recognition. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019)
32. Riesen, K., Bunke, H.: Iam graph database repository for graph based pattern recognition and machine learning. In: *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. pp. 287–297. Springer (2008)
33. Rusch, T.K., Chamberlain, B., Rowbottom, J., Mishra, S., Bronstein, M.: Graph-coupled oscillator networks. In: *International Conference on Machine Learning*. pp. 18888–18909. PMLR (2022)
34. Satorras, V.G., Hoogeboom, E., Welling, M.: E (n) equivariant graph neural networks. In: *International conference on machine learning*. pp. 9323–9332. PMLR (2021)
35. Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., Schomburg, D.: Brenda, the enzyme database: updates and major new developments. *Nucleic acids research* **32**(suppl.1), D431–D433 (2004)
36. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. *AI magazine* **29**(3), 93–93 (2008)

37. Sonthalia, R., Gilbert, A.: Tree! i am no tree! i am a low dimensional hyperbolic embedding. *Advances in Neural Information Processing Systems* **33**, 845–856 (2020)
38. Sutherland, J.J., O'brien, L.A., Weaver, D.F.: Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences* **43**(6), 1906–1915 (2003)
39. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: *International Conference on Learning Representations* (2018), <https://openreview.net/forum?id=rJXMpikCZ>
40. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: *International conference on machine learning*. pp. 6861–6871. PMLR (2019)
41. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net (2019), <https://openreview.net/forum?id=ryGs6iA5Km>
42. Yu, T., De Sa, C.: Hyla: Hyperbolic laplacian features for graph learning. *arXiv preprint arXiv:2202.06854* (2022)
43. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. *Advances in neural information processing systems* **31** (2018)
44. Zhang, T., Zheng, W., Cui, Z., Li, C.: Deep manifold-to-manifold transforming network. In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. pp. 4098–4102 (2018). <https://doi.org/10.1109/ICIP.2018.8451626>

## A Training Details

For both node and graph classification, we use grid search to tune the same set of hyperparameters for each graph architecture on the development set of a given dataset, and repeat the tuning process three times over Euclidean and hyperbolic spaces, the product of thereof, and the SPD space. We consider 5 hyperparameters: (a) learning rate  $\in \{0.1, 0.01, 0.001\}$ ; (b) dropout  $\in \{0, 0.5\}$ ; (c) weight-decay  $\in \{0, 0.005, 0.0005\}$  used to regularize a model by shrinking the L2 norm of model weights; (d) nonlinearity  $\in \{\text{TgReEig}, \text{ReEig}\}$  only used for the SPD space<sup>6</sup> and (e)  $C \in \{0.5, 0.05, 0.005, 0.0005\}$  used to control the size of SVM hyperplane. For node classification, we follow Chami et al. [11] and set batch size to a total number of graph nodes in a dataset. We train individual graph architectures for a maximum of 500 epochs, and stop training when the loss on the development set has not decreased for 200 epochs. Regarding graph classification, We set batch size to 32, and set the epochs to a maximum of 200 for training, accompanied by a patience of 100 epochs for early stopping. In all setups, we use a stack of two graph layers that perform message passing twice at each iteration.

For optimization, considering the model parameters of SPD graph neural networks are in Euclidean space, we use Adam optimizer [23] to tune these parameters, and minimize (a) the cross-entropy loss for linear classifier and (b) the multi-margin hinge loss for both SVM and centriod-based classifiers. Notably, our graph neural networks in SPD take longer time to train than the Euclidean and hyperbolic counterparts due to the requirement of computing costly eigenvalues and eigenvectors at each iteration. We choose Euclidean over Riemannian optimization, as the latter cannot guarantee the symmetry positive definiteness in SPD matrices, a potential risk incurring numerical instability.

## B Graph Architectures

Unlike GCN, GAT and Cheb, SGC and GIN both compute propagation ahead of applying feature transformation. We contrast their architectures in different spaces in Table 3.

## C Dataset Statistics

Table 4 and 5 present data statistics for the benchmark datasets in node and graph classification setups.

## D Results in Node Classification

Table 6 shows a detailed breakdown of results for node classification.

<sup>6</sup> In Euclidean and hyperbolic cases, we apply ReLU to Euclidean space and to the tangent space of hyperbolic manifold, as suggested by Chami et al. [11].

**Table 3.** Comparison of operations over two graph neural networks (SGC and GIN) in different spaces.

Operations	GNNs	Euclidean Space
Propagation	SGC GIN	$h_i^l = \sum_{j \in \mathcal{N}(i)} k_{i,j} \sum_{z \in \mathcal{N}(j)} k_{j,z} x_z^{l-1}$ $h_i^l = (1 + \epsilon) x_i^{l-1} + \sum_{j \in \mathcal{N}(i)} k_{i,j} x_j^{l-1}$
Feature Transformation	All	$p_i^l = W^l h_i^l$
Bias&Nonlinearity	All	$x_i^l = \varphi(p_i^l + b^l)$

Operations	GNNs	Hyperbolic and SPD Space
Propagation	SGC GIN	$Q_i^l = \exp(\sum_{j \in \mathcal{N}(i)} k_{i,j} \sum_{z \in \mathcal{N}(j)} k_{j,z} \log Z_z^{l-1})$ $Q_i^l = \exp((1 + \epsilon) \log Z_i^{l-1} + \sum_{j \in \mathcal{N}(i)} k_{i,j} \log Z_j^{l-1})$
Feature Transformation	All	$P_i^l = M^l \odot Q_i^{l-1}$
Bias&Nonlinearity	All	$Z_i^l = \phi(P_i^l \oplus B^l)$

**Table 4.** Dataset statistics for node classification.

Dataset	# Nodes	# Edges	# Classes	# Features
Cora	2,708	5,429	7	1,433
Citeseer	3,327	4,732	6	3,703
Pubmed	19,717	44,338	3	500
Disease	1,044	1,043	2	1,000
Airport	3,188	18,631	4	4

## E Results in Graph Classification

Table 8 shows a detailed breakdown of results for graph classification.

## F Hitchhiker’s Guide to Numerical Problems

Unlike Euclidean space, computing work in SPD does not utilize matrix multiplication, but rather relies on the use of gyrocalculus and algebraic operations in symmetric space. However, these operations may incur numerical instability. In what follows we present a list of numerical issues identified in this work, and provide solutions to address them.

*Repeated Eigenvalues.* GNNs in SPD extensively utilize logmap and expmap to switch the space of node embeddings between SPD and the space of symmetric matrices. Doing this requires the determination of the orthogonal basis and eigenvalues of an SPD matrix (or a symmetric matrix). However, when eigenvalues are not unique, the orthogonal basis is not well-defined, i.e., any vector in the eigenspace associated to a repeated eigenvalue is a legitimate eigenvector.

**Table 5.** Dataset statistics for graph classification. # Nodes and # Edges denote the number of nodes and edges on average across graphs.

Dataset	# Graphs	# Nodes	# Edges	# Classes	# Features
ENZYMES	600	32.6	124.3	6	3
PROTEINS	1,113	39.1	145.6	2	3
COX2	467	41.2	43.4	2	35
AIDS	2,000	15.6	16.2	2	38

This could incur the reproducibility issue as an orthogonal basis could be determined differently across machines, libraries, and even multiple runs. Further, the gradients of eigenvectors in the degenerate case are not well-supported in popular libraries such as Pytorch—where the gradients are stable only when the corresponding eigenvalues are unique. We address this issue by adding a random noise following a normal distribution with zero mean and the standard deviation of 0.001 to both SPD and symmetric matrices.

*Non-positive definiteness of Node Embeddings.* Machinery computing typically involves the use of imprecise floating-point systems to compute and represent numbers, particularly efficient to compute but prone to round-off and truncation errors, such as representing an irrational number approximately with a finite amount of decimals. These errors could affect the positive-definiteness of node embeddings by yielding symmetric matrices with slightly negative eigenvalues. When node embeddings are no longer in SPD, a great number of algebraic SPD operations becomes numerical unstable, such as computing the geodesic distance between two points in SPD<sup>7</sup>, mapping points from SPD to the tangent space at a given point, computing gyro-addition. To this end, we take a two-step process: (a) enabling double precision floating-point format in computing, and (b) clamping eigenvalues to ensure them being positive.

*Library Issue.* Popular libraries such as Pytorch have struggled to efficiently compute eigenvectors and eigenvalues for long. We benchmarked the running time of two factorization algorithms in Pytorch: (a) finding the eigenvectors and eigenvalues via eigenvalue decomposition and (b) finding the singular vectors and values via singular value decomposition (SVD) of 5,000 symmetric matrices with the size of  $8 \times 8$ . We found that eigenvalue decomposition was 15 times slower to run than SVD. For that reason, we utilize SVD coupled with a sign correction to compute eigenvectors and eigenvalues. We found that our solution is faster than the Pytorch in-house eigenvalue decomposition by approximately 12 times. In June 2022, Pytorch released an update that finally improved the speed of computing eigenvectors and eigenvalues.

<sup>7</sup> For instance, computing geodesic distance requires taking the square root of eigenvalues. In this case, numerical instability arises as eigenvalues are negative.

**Table 6.** Evaluation of different GNN architectures with different classifiers in the three 6-dimensional spaces, namely  $\mathbb{R}^6$ ,  $\mathbb{H}^6$ ,  $\text{SPD}(3, \mathbb{R})$ , in the node classification task. We bold the best accuracy in each row.

		$\mathbb{R}^6$	$\mathbb{H}^6$	$\text{SPD}(3, \mathbb{R})$		
		LINEAR-XE	LINEAR-XE	LINEAR-XE	SVM-MM	NC-MM
Disease	GINConv	86.4 $\pm$ 6.9	<b>95.5</b> $\pm$ 0.9	91.0 $\pm$ 3.0	95.0 $\pm$ 0.9	89.4 $\pm$ 4.6
	SGConv	87.6 $\pm$ 2.9	<b>99.9</b> $\pm$ 0.1	93.9 $\pm$ 2.2	95.8 $\pm$ 1.0	93.9 $\pm$ 1.3
	ChebConv	52.4 $\pm$ 3.9	53.5 $\pm$ 3.8	53.6 $\pm$ 3.4	58.9 $\pm$ 3.7	<b>61.4</b> $\pm$ 5.5
	GATConv	92.2 $\pm$ 8.1	<b>100.0</b> $\pm$ 0.0	98.1 $\pm$ 2.4	99.6 $\pm$ 0.8	93.7 $\pm$ 2.6
	GCNConv	88.2 $\pm$ 3.1	<b>96.9</b> $\pm$ 0.6	95.9 $\pm$ 2.1	96.0 $\pm$ 1.9	96.8 $\pm$ 1.7
Airport	GINConv	55.4 $\pm$ 3.8	65.9 $\pm$ 4.5	66.3 $\pm$ 5.1	68.6 $\pm$ 3.0	<b>74.1</b> $\pm$ 2.6
	SGConv	62.7 $\pm$ 1.6	<b>74.8</b> $\pm$ 2.8	68.1 $\pm$ 3.0	64.9 $\pm$ 3.5	71.0 $\pm$ 3.1
	ChebConv	68.7 $\pm$ 3.6	67.6 $\pm$ 2.6	74.7 $\pm$ 3.1	71.7 $\pm$ 1.6	<b>75.3</b> $\pm$ 3.2
	GATConv	65.6 $\pm$ 4.3	<b>77.0</b> $\pm$ 3.3	74.1 $\pm$ 1.5	69.8 $\pm$ 2.2	74.4 $\pm$ 2.1
	GCNConv	61.7 $\pm$ 2.2	68.6 $\pm$ 1.4	<b>71.2</b> $\pm$ 2.3	63.4 $\pm$ 3.4	71.0 $\pm$ 1.8
Pubmed	GINConv	71.2 $\pm$ 4.9	76.8 $\pm$ 1.0	<b>77.5</b> $\pm$ 0.6	77.0 $\pm$ 1.1	76.9 $\pm$ 0.8
	SGConv	70.8 $\pm$ 3.7	77.8 $\pm$ 0.8	<b>78.3</b> $\pm$ 0.5	78.1 $\pm$ 0.6	77.9 $\pm$ 0.6
	ChebConv	74.5 $\pm$ 3.6	65.1 $\pm$ 12.3	76.8 $\pm$ 3.0	<b>78.0</b> $\pm$ 0.8	77.6 $\pm$ 0.9
	GATConv	74.7 $\pm$ 3.4	77.5 $\pm$ 1.6	77.8 $\pm$ 0.6	77.2 $\pm$ 2.1	<b>78.1</b> $\pm$ 0.5
	GCNConv	77.3 $\pm$ 1.5	76.9 $\pm$ 0.5	78.0 $\pm$ 0.6	78.2 $\pm$ 0.7	<b>78.6</b> $\pm$ 0.7
Citeseer	GINConv	48.2 $\pm$ 6.3	64.1 $\pm$ 1.6	<b>68.0</b> $\pm$ 1.3	67.3 $\pm$ 1.2	67.0 $\pm$ 0.8
	SGConv	62.6 $\pm$ 3.4	64.5 $\pm$ 1.5	69.4 $\pm$ 1.0	<b>69.7</b> $\pm$ 0.8	67.9 $\pm$ 1.5
	ChebConv	63.2 $\pm$ 2.1	41.8 $\pm$ 5.8	54.6 $\pm$ 10.4	61.4 $\pm$ 4.4	<b>64.0</b> $\pm$ 2.3
	GATConv	55.0 $\pm$ 5.2	64.9 $\pm$ 1.7	67.3 $\pm$ 1.7	<b>69.2</b> $\pm$ 0.7	68.1 $\pm$ 1.1
	GCNConv	64.7 $\pm$ 2.3	64.0 $\pm$ 3.1	<b>69.9</b> $\pm$ 0.8	69.2 $\pm$ 0.8	68.2 $\pm$ 1.0
Cora	GINConv	63.2 $\pm$ 6.2	77.1 $\pm$ 1.0	<b>79.9</b> $\pm$ 0.6	79.5 $\pm$ 0.6	78.8 $\pm$ 1.0
	SGConv	75.7 $\pm$ 3.6	77.5 $\pm$ 1.5	81.5 $\pm$ 0.9	<b>81.8</b> $\pm$ 0.3	81.1 $\pm$ 0.6
	ChebConv	71.9 $\pm$ 2.8	43.9 $\pm$ 3.9	75.5 $\pm$ 3.9	77.9 $\pm$ 2.4	<b>79.2</b> $\pm$ 1.3
	GATConv	67.9 $\pm$ 4.2	78.1 $\pm$ 1.1	79.4 $\pm$ 0.9	<b>81.2</b> $\pm$ 1.4	<b>81.2</b> $\pm$ 1.1
	GCNConv	78.1 $\pm$ 1.7	78.2 $\pm$ 1.3	79.7 $\pm$ 0.9	<b>80.7</b> $\pm$ 0.5	80.2 $\pm$ 1.3

## G Classifiers

*Linear-XE.* First, we consider a linear classification layer. The layer maps an input SPD matrix  $\mathbf{Z}$  to its logarithm  $\mathbf{X} = \log(\mathbf{Z})$ , vectorizes the upper triangle elements of  $\mathbf{X}$  as  $\mathbf{x} = (X_{1,1} \cdots X_{1,d}, X_{2,2}, \cdots, X_{d,d}) \in \mathbb{R}^{d(d+1)/2}$ , then applies a linear transformation  $\mathbf{y} = \mathbf{W}\mathbf{x}$ , where  $\mathbf{W} \in \mathbb{R}^{K \times d(d+1)/2}$  is a learnable parameter matrix, and  $K$  is the number of classes. The model parameters are then trained with a standard cross entropy loss objective applied to  $\mathbf{y}$ . Finally, predictions are made after training according to  $\arg \max_k y_k$  with  $y_k$  as the  $k$ th component of  $\mathbf{y}$ .

*SVM-MM.* The second classification layer we consider is a modification of a standard SVM approach. As before in the **Linear-XE** case, we begin by mapping an input SPD matrix  $\mathbf{Z}$  to its logarithm  $\mathbf{X} = \log(\mathbf{Z})$ . Instead of vectorizing  $\mathbf{Z}$  and learning a linear transformation, in this case our implementations learns linear functionals applied directly to  $\mathbf{X}$ . This is done by identifying  $S_n$  with its dual space  $S_n^*$  via  $W \mapsto (X \mapsto \text{Tr}(WX))$ . The loss function then consists of a standard multi-class hinge loss function and a regularization term, analogous to a term proportional to  $\lambda|W|^2$  in Euclidean space. Specifically, let  $C = \frac{1}{N} \sum_{n=i}^N X_i$ , the arithmetic mean of the subset  $\{X_i\}_{i=1}^N$  in  $\text{SPD}_n$ , which is again in  $\text{SPD}_n$ .

**Table 7.** Comparison between 6- and 15-dimensional spaces across five graph neural networks and three geometries on Disease and Cora. All networks use the LINEAR-XE classifier. We bold the best accuracy in each row.

		$\mathbb{R}^6$	$\mathbb{R}^{15}$	$\mathbb{H}^6$	$\mathbb{H}^{15}$	SPD(3, $\mathbb{R}$ )	SPD(5, $\mathbb{R}$ )
Disease	GINConv	86.4 $\pm$ 6.9	91.3 $\pm$ 2.3	95.5 $\pm$ 0.9	<b>96.3</b> $\pm$ 0.8	91.0 $\pm$ 3.0	93.2 $\pm$ 2.2
	SGConv	87.6 $\pm$ 2.9	97.2 $\pm$ 1.0	<b>99.9</b> $\pm$ 0.1	<b>99.9</b> $\pm$ 0.1	93.9 $\pm$ 2.2	98.0 $\pm$ 1.0
	ChebConv	52.4 $\pm$ 3.9	51.1 $\pm$ 2.6	53.5 $\pm$ 3.8	<b>53.7</b> $\pm$ 4.8	53.6 $\pm$ 3.4	52.9 $\pm$ 4.7
	GATConv	92.2 $\pm$ 8.1	97.0 $\pm$ 3.9	<b>100.0</b> $\pm$ 0.0	<b>100.0</b> $\pm$ 0.0	98.1 $\pm$ 2.4	99.9 $\pm$ 0.2
	GCNConv	88.2 $\pm$ 3.1	89.5 $\pm$ 1.0	96.9 $\pm$ 0.6	<b>97.8</b> $\pm$ 1.2	95.9 $\pm$ 2.1	96.0 $\pm$ 3.3
Cora	GINConv	63.2 $\pm$ 6.2	74.7 $\pm$ 1.8	77.1 $\pm$ 1.9	<b>80.6</b> $\pm$ 0.8	79.9 $\pm$ 0.6	81.4 $\pm$ 0.3
	SGConv	75.7 $\pm$ 3.6	80.7 $\pm$ 0.9	77.5 $\pm$ 1.5	79.6 $\pm$ 1.3	81.5 $\pm$ 0.9	<b>82.1</b> $\pm$ 0.5
	ChebConv	71.9 $\pm$ 2.8	76.7 $\pm$ 1.3	43.9 $\pm$ 3.9	50.5 $\pm$ 3.7	75.5 $\pm$ 3.9	<b>80.3</b> $\pm$ 1.0
	GATConv	67.9 $\pm$ 4.2	74.7 $\pm$ 1.5	78.1 $\pm$ 1.1	<b>80.3</b> $\pm$ 0.5	79.4 $\pm$ 0.9	79.8 $\pm$ 1.0
	GCNConv	78.1 $\pm$ 1.7	80.8 $\pm$ 1.2	78.2 $\pm$ 1.3	80.7 $\pm$ 0.8	79.7 $\pm$ 0.9	<b>82.3</b> $\pm$ 0.5

Then the loss function we propose is

$$\begin{aligned}
& \text{Loss}_{\text{SVM-MM}}(X_1, \dots, X_N; W_1, \dots, W_K) \\
&= \lambda \sum_{k=1}^K \text{Tr}(W_k C W_k C) + \\
& \frac{1}{NK^2} \sum_{i=1}^N \sum_{k=1}^K \sum_{\substack{l=1 \\ l \neq k}}^K \max(0, 1 - \text{Tr}(W_k X_i) + \text{Tr}(W_l X_i))
\end{aligned}$$

where  $\lambda$  is a hyperparameter. Predictions are made after training according to  $\arg \max_k (X \mapsto \text{Tr}(W_k X))$ .

*NC-MM.* Lastly, we consider a centroid-based layer for classifying  $K$  classes. The layer is parametrized by  $K$  centroids  $\{\mu_k\}_{k=1}^K \subset \mathbb{R}^{d(d+1)/2}$ ,  $K$  SPD matrices  $\{P_k\}_{k=1}^K \subset \text{SPD}(d(d+1)/2, \mathbb{R})$ , and  $K$  bias terms  $\{b_k\}_{k=1}^K \subset \mathbb{R}$ . Given an input SPD matrix  $Z$ , the layer first maps  $Z$  to its logarithm  $X = \log(Z)$ , and then vectorizes the upper triangle elements of  $X$  as  $x = (X_{1,1} \cdots X_{1,d}, X_{2,2}, \dots, X_{d,d}) \in \mathbb{R}^{d(d+1)/2}$ . A centroid-based similarity value for each class  $k$  is then computed as  $\text{sim}(x, \mu_k) := -\frac{1}{2}(x - \mu_k)P_k(x - \mu_k)^\top + b_k$ . The model parameters are trained using a standard multi-hinge loss objective applied to the feature vector  $(\text{sim}(x, \mu_k))_{k=1}^K$ . Finally, predictions are made after training according to  $\arg \max_k \text{sim}(x, \mu_k)$ .

## H Product Manifold

Let  $M_1, M, \dots, M_k$  be a sequence of smooth manifolds. The product manifold is given by the Cartesian product  $M = M_1 \times M_2 \times \cdots \times M_k$ . Each point  $p \in M$  has the coordinates  $p = (p_1, \dots, p_k)$ , with  $p_i \in M_i$  for all  $i$ . Similarly, a tangent vector  $v \in T_p M$  can be written as  $(v_1, \dots, v_k)$ , with each  $v_i \in T_{p_i} M_i$ . If each  $M_i$  is equipped with a Riemannian metric  $g_i$ , then the product manifold  $M$  can be given the product metric where  $g(v, w) = \sum_{i=1}^k g_i(v_i, w_i)$ .

**Table 8.** Evaluation of different GNN architectures with different classifiers in the three 36-dimensional spaces, namely  $\mathbb{R}^{36}$ ,  $\mathbb{H}^{36}$ ,  $\text{SPD}(8, \mathbb{R})$ , in the graph classification task. We bold the best accuracy in each row.

		$\mathbb{R}^{36}$	$\mathbb{H}^{36}$	$\text{SPD}(8, \mathbb{R})$		
		LINEAR-XE	LINEAR-XE	LINEAR-XE	SVM-MM	NC-MM
COX2	GINConv	80.0 $\pm$ 2.5	79.8 $\pm$ 2.1	81.7 $\pm$ 1.9	78.7 $\pm$ 2.5	<b>82.3</b> $\pm$ 3.0
	SGConv	78.3 $\pm$ 1.8	78.1 $\pm$ 4.0	77.9 $\pm$ 2.5	75.1 $\pm$ 1.9	<b>79.8</b> $\pm$ 3.5
	ChebConv	84.5 $\pm$ 1.3	75.1 $\pm$ 1.3	79.6 $\pm$ 3.5	78.7 $\pm$ 3.5	<b>85.3</b> $\pm$ 2.9
	GATConv	79.6 $\pm$ 3.1	77.0 $\pm$ 3.7	79.1 $\pm$ 2.0	78.7 $\pm$ 0.0	<b>81.7</b> $\pm$ 2.5
	GCNConv	79.1 $\pm$ 2.8	81.1 $\pm$ 1.4	<b>84.3</b> $\pm$ 1.9	74.9 $\pm$ 1.2	77.7 $\pm$ 4.0
AIDS	GINConv	85.9 $\pm$ 1.3	89.7 $\pm$ 0.6	<b>95.8</b> $\pm$ 1.2	92.3 $\pm$ 1.4	92.3 $\pm$ 2.6
	SGConv	77.1 $\pm$ 0.8	84.2 $\pm$ 1.5	<b>97.7</b> $\pm$ 0.9	94.0 $\pm$ 1.1	92.5 $\pm$ 0.7
	ChebConv	88.3 $\pm$ 5.8	86.0 $\pm$ 1.7	<b>94.6</b> $\pm$ 1.0	93.0 $\pm$ 4.6	94.4 $\pm$ 1.6
	GATConv	77.2 $\pm$ 2.2	84.2 $\pm$ 4.0	<b>95.0</b> $\pm$ 1.0	93.2 $\pm$ 4.1	94.2 $\pm$ 0.7
	GCNConv	77.0 $\pm$ 0.5	89.2 $\pm$ 1.2	<b>96.2</b> $\pm$ 0.4	93.1 $\pm$ 1.8	91.8 $\pm$ 0.8
ENZYMES	GINConv	<b>31.8</b> $\pm$ 4.8	27.7 $\pm$ 5.9	27.2 $\pm$ 4.1	29.3 $\pm$ 6.9	31.2 $\pm$ 6.1
	SGConv	28.7 $\pm$ 4.7	28.0 $\pm$ 3.2	39.5 $\pm$ 5.1	37.2 $\pm$ 4.4	<b>41.5</b> $\pm$ 2.6
	ChebConv	34.5 $\pm$ 4.2	26.5 $\pm$ 3.0	35.0 $\pm$ 2.2	38.8 $\pm$ 5.3	<b>40.0</b> $\pm$ 7.4
	GATConv	<b>31.8</b> $\pm$ 5.4	29.2 $\pm$ 3.8	30.0 $\pm$ 3.1	29.3 $\pm$ 3.2	29.5 $\pm$ 5.3
	GCNConv	30.2 $\pm$ 5.7	28.2 $\pm$ 3.3	33.2 $\pm$ 2.9	29.0 $\pm$ 1.1	<b>39.2</b> $\pm$ 3.2
PROTEINS	GINConv	70.8 $\pm$ 1.2	71.5 $\pm$ 0.8	<b>74.4</b> $\pm$ 1.2	72.3 $\pm$ 2.4	73.0 $\pm$ 1.6
	SGConv	72.0 $\pm$ 1.0	72.0 $\pm$ 2.5	75.3 $\pm$ 1.7	74.6 $\pm$ 1.5	<b>77.1</b> $\pm$ 1.3
	ChebConv	68.6 $\pm$ 1.1	72.3 $\pm$ 0.9	73.9 $\pm$ 2.0	<b>75.1</b> $\pm$ 1.6	74.9 $\pm$ 1.6
	GATConv	70.6 $\pm$ 0.9	72.0 $\pm$ 3.3	72.0 $\pm$ 2.9	72.7 $\pm$ 3.2	<b>72.8</b> $\pm$ 1.5
	GCNConv	71.8 $\pm$ 1.2	72.8 $\pm$ 1.2	74.7 $\pm$ 0.9	74.3 $\pm$ 1.2	<b>75.2</b> $\pm$ 1.1

*Application to Graph Neural Networks.* We construct graph convolutional network (GCN) in product manifold given by  $M = \mathbb{H}^m \times \mathbb{R}^m$ . Let  $Z_i^l = (Z_{i,1}^l, Z_{i,2}^l) \in M$  be the embedding of the node  $i$  at the  $l$  layer, with  $Z_{i,1}^l \in \mathbb{H}^m$  and  $Z_{i,2}^l \in \mathbb{R}^m$ . In the following, we implement three building blocks that enable GCN to operate in product manifold.

- *Feature transformation* combines hyperbolic and linear maps to transform each point  $Z_i^{l-1}$  in  $M$  at each layer  $l$ , denoted by:

$$Q_i^l = W^l \otimes Z_i^{l-1} = \begin{pmatrix} W_{11}^l & W_{12}^l \\ W_{21}^l & W_{22}^l \end{pmatrix} \otimes \begin{pmatrix} Z_{i,1}^{l-1} \\ Z_{i,2}^{l-1} \end{pmatrix} = \begin{pmatrix} (W_{11}^l \odot Z_{i,1}^{l-1}) \oplus (W_{12}^l \odot \exp(Z_{i,2}^{l-1})) \\ W_{21}^l \log(Z_{i,1}^{l-1}) + W_{22}^l Z_{i,2}^{l-1} \end{pmatrix}$$

Where  $W_{ij}^l \in GL(m, \mathbb{R})$  for all  $i, j$  (with learnable weights adjusted to match training data),  $Q_i^l = (Q_{i,1}^l, Q_{i,2}^l) \in \mathbb{H}^m \times \mathbb{R}^m$ ,  $\oplus$  is Möbius addition,  $\exp$  and  $\log$  are the mappings between hyperbolic space and its tangent space, and  $\odot$  is hyperbolic matrix multiplication, denoted by  $W_{11}^l \odot Z_{i,1}^{l-1} := \exp(W_{11}^l \log(Z_{i,1}^{l-1}))$  for example.

- *Propagation* aggregates information from all the neighbors  $\mathcal{N}(i)$  of a given node  $i$ , with  $k_{i,j}$  as the weight for node  $i$  and  $j$ :

$$P_i^l = (\exp(\sum_{j \in \mathcal{N}(i)} k_{i,j} \log(Q_{j,1}^l)), \sum_{j \in \mathcal{N}(i)} k_{i,j} Q_{j,2}^l)$$



Where  $P_i^l = (P_{i,1}^l, P_{i,2}^l) \in \mathbb{H}^m \times \mathbb{R}^m$ , and  $k_{i,j} = c_i^{-\frac{1}{2}} c_j^{-\frac{1}{2}}$  with  $c_i$  as the cardinality of  $\mathcal{N}(i)$ .

- *Bias and Non-linearity* adds the biases to the propagation results followed by applying two non-linear functions:

$$Z_i^l = (\varphi_1(P_{i,1}^l \oplus B_1^l), \varphi_2(P_{i,2}^l + B_2^l))$$

Where  $\varphi_1 : x \mapsto \exp(\text{ReLU}(\log(x)))$ ,  $\varphi_2 : x \mapsto \text{ReLU}(x)$ ,  $\oplus$  is Möbius addition, and  $B_1^l \in \mathbb{H}^m$  and  $B_2^l \in \mathbb{R}^m$  are learnable bias weights.

*Results.* Table 9 compares product space with SPD, Euclidean and hyperbolic spaces. Results show that the product space  $\mathbb{H}^3 \times \mathbb{R}^3$  considerably underperforms  $\text{SPD}_3$ , despite both combining Euclidean and hyperbolic subspaces. Furthermore, the product space performs worse than the Euclidean and hyperbolic counterparts ( $\mathbb{R}^6$  and  $\mathbb{H}^6$ ) on non-hyperbolic datasets including Pubmed, Citeseer and Cora.

This matches the mathematical expectation that SPD, which combines Euclidean and hyperbolic features in a more intricate way than product manifolds, has a higher representation capacity, even though both classes of spaces exhibit mixed curvature. For that reason, SPD more readily accommodates a wider variety of graph data.

**Table 9.** Comparison to the product space ( $\mathbb{H}^3 \times \mathbb{R}^3$ ) for node classification. Results are from GCNConv coupled with Linear-XE.

	Disease $\delta = 0$	Airport $\delta = 1$	Pubmed $\delta = 3.5$	Citeseer $\delta = 5$	Cora $\delta = 11$
$\mathbb{R}^6$	88.2 $\pm$ 3.1	61.7 $\pm$ 2.2	77.3 $\pm$ 1.5	64.7 $\pm$ 2.3	78.1 $\pm$ 1.7
$\mathbb{H}^6$	<b>96.9</b> $\pm$ 0.6	68.6 $\pm$ 1.4	76.9 $\pm$ 0.5	64.0 $\pm$ 3.1	78.2 $\pm$ 1.3
$\text{SPD}_3$	95.9 $\pm$ 2.1	<b>71.2</b> $\pm$ 2.3	<b>78.0</b> $\pm$ 0.6	<b>69.9</b> $\pm$ 0.8	<b>79.7</b> $\pm$ 0.9
$\mathbb{H}^3 \times \mathbb{R}^3$	95.9 $\pm$ 0.9	68.1 $\pm$ 4.7	74.7 $\pm$ 1.9	62.2 $\pm$ 1.7	71.8 $\pm$ 4.6